Clube Delphi + PHP

DevMedia group

Ano 9 · Edição 105 · R$11,90

Borland Socket Server

Quer aplicações rápidas e robustas rodando na Web? BSS nelas!

QuickUpdate
TList<T>; function Add(const Value: T): Integer; Isso é Delphi?

Delphi Prism
Desenvolva aplicações .net com Delphi Prism

Boas Práticas
Conheça os padrões Command, Interpreter, Memento entre outros

Boa Idéia
Faça um controle de cheques usando orientação a objetos

EasyDelphi
Conhecendo e utilizando os componentes Ribbon Controls

Delphi for PHP
Utilize Ajax em aplicações Delphi for PHP 2.0

ISSN 1517990-7

9 771517 990009 00105

Na Web :

Confira no portal ClubeDelphi PLUS 4 vídeo-aulas:

- Novidades da IDE do Delphi 2009
- Como utilizar Ajax em projetos PHP utilizando Delphi for PHP
- Conhecendo o FastMM
- Formatando código fonte com DelFor Exp

Engenharia de Software
CONFERENCE

22 e 23 de maio – São Paulo

Raras são as oportunidades de ter acesso a informações
que podem transformar sua carreira. Essa é uma delas.

Reserve sua agenda. Inscrições limitadas.
www.devmedia.com.br/es_conference

Maiores informações através do e-mail evento@devmedia.com.br
ou pelo telefone (21) 3382-5025

DevMedia group

Sumário

Olá, eu sou o **DevMan**! Desta página em diante, eu estarei lhe ajudando a compreender com ainda mais facilidade o conteúdo desta edição. Será um prazer contar com sua companhia! Confira abaixo o que teremos nesta revista:

Delphi — Tutorial

10 – Aplicações multicamadas com BSS
Criando sistemas multicamadas utilizando Borland Socket Server - Parte I
[*Daniel Sobrinho Laporte*]

Delphi — Boas Práticas, Projeto/Análise

18 – Padrões de Projeto
Conheça os padrões Command, Interpreter, Memento entre outros
[*Paulo Quicoli*]

Boas Práticas — Boa Idéia

24 – tiOPF na Prática
Faça um controle de cheques usando orientação a objetos
[*Paulo Quicoli*]

Easy Delphi — Easy Delphi

32 – Introdução ao Ribbon Control do Delphi 2009
Conheça os novos componentes da VCL
[*Fabrício Hissao*]

PHP — PHP

40 – Delphi for PHP x Ajax
Utilize Ajax em aplicações Delphi for PHP 2.0
[*Adriano Santos*]

Novidades

46 – Delphi Prism
Desenvolva aplicações .net com o novo Delphi Prism
[*Adriano Santos*]

Você percebeu os ícones ao lado de cada matéria? Eles indicam o que você vai encontrar no artigo – dessa forma, você também pode ter uma idéia geral do que vai encontrar nesta edição como um todo! Os editores trabalham sempre no sentido de fechar a revista seguindo esta definição, para oferecer a você o melhor conteúdo didático!

Confira abaixo a lista com a definição dos tipos de artigo encontrados nesta edição:

[Boas Práticas] Um dos objetivos da revista é levar para o leitor não somente as melhores técnicas mas também as melhores práticas. Esse tipo de artigo foca em técnicas que poderão aumentar a qualidade do desenvolvimento de software.

[Boa Idéia] A esse tipo de artigo é quase um artigo "Mão na Massa" - a diferença é que o exemplo do artigo é tão criativo que é considerado uma "Boa Idéia", dentro do contexto do desenvolvimento de software.

[Expert] Artigo com foco no leitor avançado.

[Novidades] Artigos sobre tecnologias de ponta, que ainda não fazem parte o dia-a-dia do desenvolvedor mas que prometem ser a próxima febre.

[Easy Delphi] Com foco no desenvolvedor iniciante.

[PHP] Artigo com foco no PHP puro.

Clube Delphi +PHP

Ano 9 - 105ª Edição 2009 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Editor Geral
Adriano Santos
adriano.santos@devmedia.com.br

Editor Técnico
Paulo Quicoli
pauloquicoli@devmedia.com.br

Comissão Editorial
Fabrício Desbessel, Paulo Quicoli, Adriano Santos e Guinther Pauli.

Diagramação
Romulo Araujo
romulo@devmedia.com.br

Revisão e Supervisão
Thiago Vincenzo - thiago.v.ciancio@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor
A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Carmelita Mulin – Atendimento ao Leitor
www.devmedia.com.br/central/default.asp
(21) 3382-5025

Kaline Dolabella – Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 3382-5025

Publicidade
Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

Fale com o Editor!
É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Adriano Santos - Editor da Revista
adriano.santos@devmedia.com.br

EDITORIAL

Essa edição chegou às bancas com cheirinho de novidade no ar. E é porque temos muitas novidades mesmo para deixar o leitor ainda mais curioso e empolgado com a revista.

Esse mês nossa matéria de capa fala sobre uma tecnologia extremamente importante nos dias de hoje: multi-camadas. Assunto em pauta até na hora do cafezinho, tenho certeza disso. Posso arriscar a dizer que os dois assuntos mais importantes do mundo de TI hoje resumem-se em Multi-Camadas e Orientação a Objetos. Isso acontece porque cada vez mais precisamos de velocidade e praticidade na hora de desenvolver ou distribuir nossas aplicações. Eu trabalhei e trabalho ainda em projetos totalmente multi-camadas e outros nem tanto. Pude ver de perto a diferença de velocidade entre uma aplicação comum e outra n-tier. De fato é o que motiva cada vez mais a produzir aplicações com essa tecnologia. Esse mês quem vai falar especificamente desse assunto é o nosso novo colunista Daniel Sobrinho Laporte. Ele convive diariamente com aplicações que utilizam o BSS (Borland Socket Server) e pode nos dar uma aula de velocidade de acesso e robustez de aplicação. Em seu mini-curso em duas etapas, mostrará como montar um servidor de aplicações e as versões cliente desse sistema.

O Paulo continua e finaliza sua série de artigos sobre padrões de projeto, dessa vez falando e demonstrando especificamente dos padrões Command, Interpreter e Memento. Veja com ele como implementar esse padrões em sua aplicação. Na seqüência o Paulo fala sobre orientação a objetos e mostra como criar um aplicativo para controle de cheques.

Na sessão EasyDelphi o Fabrício, também novo na equipe, fala sobre os novíssimos controles Ribbon Controls existentes no RAD Studio Delphi 2009. Com eles é possível criar aplicações com a aparência do pacote Microsoft Office 2007.

Ajax está na moda, não é para menos. As páginas que fazem uso dessa tecnologia ficam realmente mais inteligentes e rápidas. Mas como aplicar Ajax em Delphi for PHP? Veremos isso em meu artigo.

.net virou mania mundial, e como desenvolver hoje em .net com Delphi utilizando Framework mais avançado que existe? A solução é Delphi Prism. Ele veio para ficar e certamente trará bons frutos. Com ele é possível programar em Delphi.net dentro do Shell do Visual Studio 2008. Isso mesmo, veremos que o Delphi Prism na verdade virou um plug-in que pode ser instalado com ou sem o Visual Studio instalado.

Pessoal, é isso que tenho a dizer. Muita leitura e código.

Adriano Santos

adriano.santos@devmedia.com.br

Editor Chefe da revista Clube Delphi
Conheça nosso BLOG em *clubedelphiphp.blogspot.com*

ClubeDelphi PLUS

DevMedia group

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS.
Para mais informações sobre o pacote PLUS, acesse:
http://www.devmedia.com.br/clubedelphi/portal.asp

Hospede com quem fala sua língua!

Local host

Hospedagem Profissional
Revenda
Parcerias

Ativação imediata!

Faça sua assinatura, tenha 30 dias grátis

e seu primeiro boleto para 60 dias após assinatura em qualquer plano

UNIX I

a partir R$ 28,00

- 1 GB Espaço em disco
- 60 GB Transferência
- 30 Contas de e-mail
- 10 Domínios
- Painel de Controle em Português
- Atendimento exclusivo

Contratar

UNIX II

a partir R$ 58,00

- 3 GB Espaço em disco
- 120 GB Transferência
- 60 Contas de e-mail
- 15 Domínios
- Painel de Controle em Português
- Atendimento exclusivo

Contratar

UNIX III

a partir R$ 98,00

- 6 GB Espaço em disco
- 300 GB Transferência
- 90 Contas de e-mail
- 20 Domínios
- Painel de Controle em Português
- Atendimento exclusivo

Contratar

R$ 5,90 — Mini html

Tráfego 1 GB
Espaço 100 MB
3 C.Ppostal, Espaço 3G
Domínio 3

R$ 15,90 — html

Tráfego 25 GB
Espaço 500 MB
20 C.Ppostal, Espaço 20G
Domínio Ilimitado

R$ 15,90 — ASP

Tráfego 25 GB
Espaço 500 MB
20 C.Ppostal, Espaço 20G
Domínio Ilimitado
MySql 50m

R$ 20,90 — PHP

Tráfego 25 GB
Espaço 500 MB
20 C.Ppostal, Espaço 20G
Domínio Ilimitado
MySql 50m

Formato exclusivo de trabalho para empresas de desenvolvimento de sites, websites, portais, serviços e negócios que necessitam de grandes tráfegos, monitoramento e backup.

www.lhost.com.br

assinar@lhost.com.br

ClubeDelphi PLUS

Informativo ClubeDelphi

Portal ClubeDelphi

www.clubedelphi.net/portal

+ 600 vídeo aulas e 7 cursos online

Brinde na web desta edição - 4 vídeos! Confira:

4 Vídeos

• **Novidades da IDE do Delphi 2009:** Todos os principais recursos da IDE serão mostrados neste vídeo. Project Manager, Class Explorer, Resource Manager e Build Configuration são alguns dos tópicos abordados.
http://www.devmedia.com.br/articles/viewcomp.asp?comp=11961

• **Como utilizar Ajax em projetos PHP utilizando Delphi for PHP**
http://www.devmedia.com.br/articles/viewcomp.asp?comp=10971&hl=

• **Conhecendo o FastMM**
http://www.devmedia.com.br/articles/viewcomp.asp?comp=6173&hl=

• **Formatando código fonte com DelFor Exp**
http://www.devmedia.com.br/articles/viewcomp.asp?comp=5454&hl=

Para acessar os vídeos utilize os seguintes dados
Login: **DVM.PLS** *Senha:* **kv28013**

Gostou das vídeo aulas? O portal *www.devmedia.com.br* possui mais de **2 mil vídeo aulas** e **dezenas de cursos online** sobre desenvolvimento de software! Agora você pode comprar as vídeo aulas que preferir e fazer sua própria combinação de vídeos! Saiba mais em *www.devmedia.com.br/creditos*

Edições Anteriores da ClubeDelphi

Você pode comprar todas as edições anteriores da ClubeDelphi através do site da DevMedia!

Para isso basta acessar *https://seguro.devmedia.com.br/edicoes_anteriores.asp*

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Para votar, você vai precisar do código e senha de banca desta edição, que é:
Login: **DVM.PLS** Senha: **kv28013**

Hospede seus sites com os profissionais da Hospedix.

3 DOMÍNIOS PRINCIPAIS

200 DOMÍNIOS HOSPEDADOS

100GB DE TRANSFERÊNCIA MENSAL

30GB PARA E-MAILS

2GB DE ESPAÇO EM DISCO

Clientes Hospedix ainda podem optar por uma de nossas promoções:

1 registro de domínio (.com .net .org ou .info) grátis e isento de taxas anuais.

ou

1 registro de domínio (.com.br) por apenas R$15,00 anuais enquanto for cliente.

Tudo isso por apenas **R$ 25,90** por mês

Assine agora mesmo e descubra por que Hospedix é a hospedagem de sites preferida entre os profissionais.

HOSPEDIX
HOSPEDAGEM PROFISSIONAL
www.hospedix.com.br

30 DIAS GRÁTIS

Ao assinar, digite o código de desconto exclusivo para leitores desta revista e ganhe 30 dias inteiramente grátis.

RVSTMEDIA

Quick Update

TList<T>; function Add(const Value:T):Integer; Isso é Delphi?

Paulo Quicoli

pauloquicoli@devmedia.com.br

Editor técnico da revista ClubeDelphi e .Net Maganize, formado em processamento de dados pela FATEC-TQ. Atua como analista e desenvolvedor da ControlM Informática (www.controlm.com.br). Utiliza Delphi desde sua primeira versão, entusiasta do desenvolvimento orientado a objetos tem publicado vários artigos sobre o assunto.Blog: www.pauloquicoli.spaces.live.com

O título acima talvez possa lhe ter gerado alguma confusão, mas a resposta é *sim*, isso é *Delphi*. Na sua última versão o *Delphi* introduziu alguns novos recursos na linguagem. Um deles é chamado *Generics*. O desenvolvedor *Delphi* que já utilizou o *.NET Framework 2* já está familiarizado com essa sintaxe, se não estiver, pelo menos já a viu em algum código. Essa é a notação do *Generics* no *Delphi*. Vamos entender o que é tudo isso.

O que é Generics?

Acredito que fazer uma explicação formal aqui complicaria mais então vou explicar de forma prática. O exemplo mais comum para se entender o poder do seu uso é utilizar um *TList*. Até a versão *2007* do *Delphi Win32* se fosse necessário utilizar um *TList* para armazenar e recuperar uma lista de objetos do tipo *TEdit* faríamos como na **Listagem 1**.

Listagem 1. Utilizando um TList

```
01 var
02   edit: TEdit;
03   ListaEdits: TList;
04 begin
05   edit := TEdit.Create(self);
06
07   ListaEdits := TList.Create;
08   ListaEdits.Add(edit);
09   TEdit(ListaEdits[0]).Name := 'Teste';
```

Observe que para acessar o elemento recém inserido em *ListaEdits* é preciso realizar um *cast*. Isso porque o *TList* aceita tipos *Pointer*. Essa operação de *cast* que realizamos para *transformar* de *Pointer* para *TEdit* também é chamada de *unboxing*. Quando adicionamos o *TEdit* na lista foi realizado um *boxing*. Para entender isso posso dizer que quando o *TList* recebe o *TEdit*, ele é *transformado – boxing* - em um *TPointer* e depois quando vamos utilizá-lo o *transformamos* novamente – *unboxing* – em *TEdit*.

O mesmo código utilizando *generics* é exibido na **Listagem 2**.

Veja que a declaração do *TList* mudou. Agora temos um <*TEdit*>. Isso indica que o *TList* é *tipado de forma genérica*, fazendo com que internamente ele não armazene *Pointer*, mas sim *TEdit*. Dessa forma veja que não foi necessário realizar um *cast* para acessar a propriedade *Name* do elemento 0. Vamos dar uma olhadinha na estrutura desse *TList* genérico, veja **Listagem 3**.

Listagem 2. Utilizando um TList genérico

```
01 var
02   edit: TEdit;
03   ListaEdits: TList<TEdit>;
04 begin
05   edit := TEdit.Create(self);
06
07   ListaEdits := TList<TEdit>.Create;
08   ListaEdits.Add(edit);
09   ListaEdits[0].Name := 'Teste';
```

Listagem 3. Listagem parcial da classe TList<T>

```
01 TList<T> = class(TEnumerable<T>)
02 private
03    FItems: array of T;
04    FCount: Integer;
05 //...
06 public
07 //...
08  function Add(const Value: T): Integer;
09  procedure Insert(Index: Integer; const Value: T);
10  procedure Sort(const AComparer: IComparer<T>); overload;
11 //..
12 end;
```

Com certeza você deve estar se perguntando agora o que é esse *<T>*. Ele é um parâmetro genérico, que aponta não para uma referência, mas para um tipo. Dessa forma se eu quisesse que meu *TList* aceitasse apenas números inteiros, eu o definiria como *TList<Integer>*. Se eu tentar adicionar outro tipo de dado a esse *TList*, o compilador dispara um erro.

Vantagens

Se compararmos a **Listagem 1** com a **Listagem 2** podemos ver que não precisamos mais realizar *cast*. O que, por conseqüência, deixa o código mais legível. Outro ponto importante é a segurança. Como temos um *TList<TEdit>*, só poderemos adicionar e recuperar deles objetos do tipo *TEdit*. Assim, a possibilidade de você passar um *TPointer* no lugar de um *TEdit* é praticamente nula, pois o compilador irá verificar isso.

Herança e Generics

Na orientação a objetos, no que diz respeito à herança, podemos atribuir uma instância de uma classe filha para uma variável do tipo pai. Contudo o inverso não é possível. Para exemplificar isso veja o código da **Listagem 4**.

O código compila porque *TList* herda de *TObject*, portando podemos atribuir sua instância à variável *obj* do tipo *TObject*. Agora, considere o código da **Listagem 5**.

Listagem 4. Aplicando herança

```
01 var
02   obj: TObject;
03   Lista: TList;
04 begin
05   Lista := TList.Create;
06   obj := Lista;
07 end;
```

Listagem 5. Aplicando herança em generics

```
01 var
02   ListaComponent: TList<TComponent>;
03   ListaEdit: TList<TEdit>;
04   ListaObject: TList<TObject>;
05 begin
06   ListaComponent := ListaEdit;
07   ListaObject := ListaComponent;
08 end;
```

Se seguirmos o raciocínio da herança, não há nada de errado com a **Listagem 5**. Porém não é isso que o compilador nos mostra. Veja **Figura 1**.

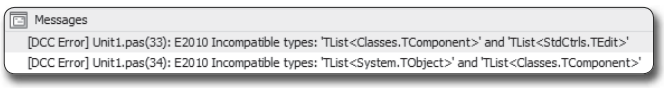


Figura 1. Erro de compilação

O compilador nos alerta que não podemos fazer atribuição de tipos incompatíveis. Isso significa que classes genéricas, mesmo sendo tipadas para classes relativas – como *TComponent* e *TEdit* – não permitem aplicar herança dessa forma.

O uso do recurso de *generics* no *Delphi* não é limitado apenas ao que foi mostrado aqui. O próprio *Delphi* já disponibiliza classes genéricas, como o *TList<T>*. Além disso, podemos ter métodos, classes, *records* genéricos criados por nós mesmos. Espero que o interesse pelo assunto tenha sido despertado. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos intermediários sobre Delphi Win 32 e Delphi for .NET

Aplicações multicamadas com BSS

Criando sistemas multicamadas utilizando Borland Socket Server - Parte I

Resumo do DevMan

Sistemas multicamadas têm sido amplamente discutidos nas comunidades de tecnologia, no entanto, muitos desenvolvedores ainda possuem receios em migrar seus softwares cliente-servidor para esta arquitetura ou iniciar aplicações deste tipo, devido à falta de conhecimento ou até mesmo por achar a implementação muito complexa.

Neste artigo veremos como desenvolver um sistema em multicamadas utilizando as tecnologias DataSnap e BSS (Borland Socket Server), abordando as principais maneiras para a construção do ambiente, além dos principais conceitos para o desenvolvimento de uma aplicação com qualidade.

Nesse artigo veremos

- Criação do banco de dados de modelo;
- Criação de uma solução multicamadas de alta escalabilidade;
- Configuração do ambiente de desenvolvimento;
- Entendimento da arquitetura, tecnologia e dicas para implementação;

Qual a finalidade

Desenvolver qualquer tipo de aplicação que permita acesso pela internet, unida a alta performance, sendo aplicável a qualquer segmento de software.

Quais situações utilizam esse recurso?

Qualquer ambiente que necessite de grande disponibilidade, empresas que possuam sistemas complexos ou integração entre matrizes e filiais.

Daniel Sobrinho Laporte
daniel.laporte@gmail.com
É graduado em sistemas de informação e pós-graduado em administração de bancos de dados pela Universidade de Franca. É analista de sistemas e desenvolvedor Delphi desde 2005 desenvolvendo aplicações cliente/servidor e multicamadas em Delphi com Firebird.

A cada dia o crescimento de usuários que navegam na internet se torna maior, isto se deve ao fato de que este meio de comunicação nunca esteve tão acessível a todos, devido às inúmeras opções de provedores que oferecem os mais variados planos. Claro que com tanta facilidade, os usuários deste serviço querem cada vez mais disponibilidade para desfrutar da web. Como no cenário empresarial não poderia ser diferente, cada vez mais as empresas de tecnologia têm investido grandes esforços para disponibilizar

suas aplicações e serviços de forma performática, produtiva e atendendo as características que cada vez mais tem atraído os usuários de sistemas: a possibilidade de acessar seu software de qualquer lugar, e a qualquer momento.

Quando necessitamos fazer uma solução deste porte, a primeira situação que nos vem à cabeça é que devemos desenvolver algo totalmente voltado para *web*, mas veremos que podemos desenvolver soluções *win32* que atendam esses requisitos com êxito.

Neste artigo iremos utilizar a tecnologia *DataSnap* para a construção e integração das nossas aplicações cliente e servidora, juntamente com o *BSS (Borland Socket Server)* para realizarmos a comunicação entre as camadas.

Assim sendo, vamos criar uma aplicação simples na qual simule o empréstimo de livros de uma biblioteca, utilizando todas estas tecnologias citadas e ainda aplicando algumas práticas que podem elevar a produtividade do desenvolvimento. Antes de começarmos a desenvolver é fundamental conhecer como será o comportamento do nosso ambiente. Após uma breve explicação sobre este modelo iremos dar início a criação do nosso banco de dados de exemplo.

O modelo multicamadas

Geralmente ouvimos tanto falar nos termos três camadas quanto em multicamadas, na verdade, ambos são a mesma coisa, porém três camadas refere-se a divisão comum que compõe qualquer tipo de aplicação deste escopo, sendo banco de dados, aplicação cliente e aplicação servidora, mais DLL's, bibliotecas de funções, ou até mesmo outras aplicações servidoras podem ser consideradas camadas extras.

A receita do bolo trata três quesitos básicos para o desenrolar do projeto. O primeiro consiste no banco de dados. O aconselhável neste tipo de ambiente é que seu *banco de dados* possua o mínimo possível de objetos (*procedures*, *triggers*, *views*), e tarefas realizadas dentro de si, para que assim seja fácil tornar seu servidor *multi-bancos* sem grandes ajustes.

O próximo quesito é o *servidor de aplicação*, este concentra todas as regras de negócios, instruções *SQL*, todas as tarefas pesadas e complexas são delegadas a ele.

E por último, a *aplicação cliente* que é composta pela camada de apresentação dos dados para o usuário, e não possuem vínculo direto com o *banco de dados*, sendo telas realizando requisições ao servidor de aplicação e contendo algumas validações simples a nível de consistência, para que os dados sejam aplicados ao banco de dados, separando as regras de negócios das regras de consistência.

Todo este canal de comunicação entre as camadas é realizado pelo *BSS (Borland Socket Server)*, que nada mais é do que uma tecnologia baseada em *DCOM (Distributed COM)*, funcionando por meio de *Sockets* que diferente de *SOAP* que se comunicam através de *XML*, os *Sockets* comunicam-se por meio de protocolos *TCP* ou *UDP*.

Na **Figura 1** podemos ver uma ilustração de como é realizada a comunicação entre todo o ambiente. Após esta explicação vamos dar início a modelagem e criação do nosso banco de dados.

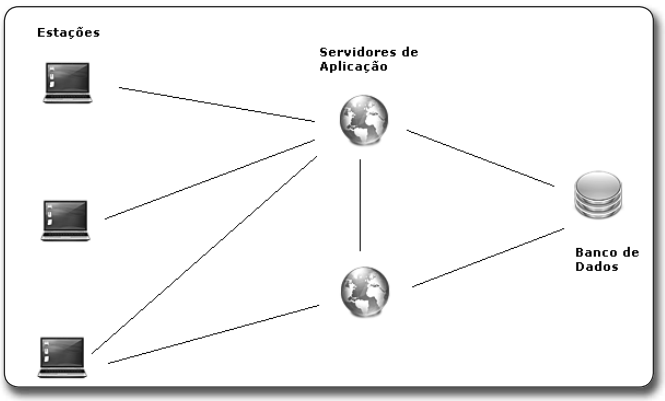


Figura 1. Exemplo da interação entre as camadas

Nota

É importante atentarmos para diferença de regras de negócios e regras de consistência. As regras de negócios são as tarefas diretamente relacionadas à persistência dos dados, enquanto as regras de consistência são responsáveis pelas validações simples na camada de apresentação, fazendo com que os dados sejam enviados ao servidor de aplicação já validados e prontos para serem salvos.

Criação do banco de dados

A **Figura 2** mostra o relacionamento entre as tabelas do nosso sistema de bibliotecas, que basicamente é composto por pessoas, editoras, livros e seus itens. A explicação para a tabela de itens de livros se deve ao fato de que um mesmo livro possui várias cópias, e cada uma delas um código que identificará sua disponibilidade, e por último a tabela de empréstimos com seus itens.

Para o exemplo, utilizarei o banco de dados *Firebird 1.5*, porém, o que será mostrado poderá ser aplicado ou adaptado a qualquer outro banco dados em uma versão de sua preferência. A **Figura 2** apresenta a modelagem sugerida para o nosso sistema, e em seguida os scripts para sua criação conforme a **Listagem 1**. Para a execução dos scripts utilize um gerenciador como *IBOConsole* ou *IBExpert*.

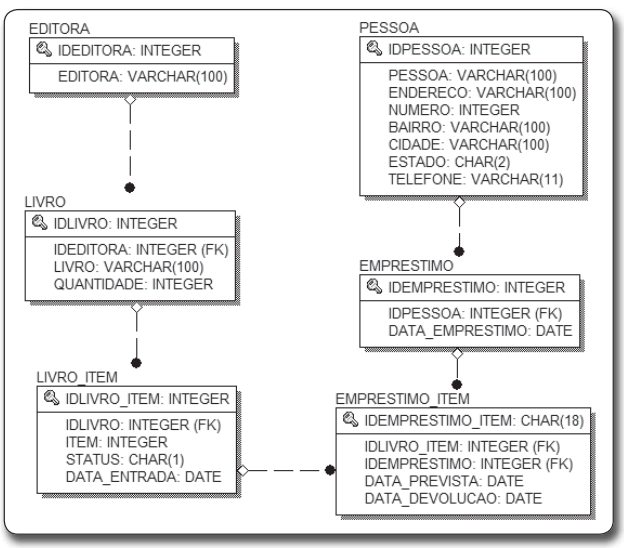


Figura 2. Modelagem de dados do sistema

Listagem 1. Script para a criação do banco de dados

```
SET SQL DIALECT 3;
SET NAMES WIN1252;

SET CLIENTLIB '<diretório_do_banco>\fbclient.dll';

CREATE DATABASE '<diretório_do_banco>\BIBLIOTECA.FDB'
USER 'SYSDBA' PASSWORD 'masterkey'
PAGE_SIZE 4096

DEFAULT CHARACTER SET WIN1252;

CREATE GENERATOR GEN_IDEDITORA;

CREATE GENERATOR GEN_IDLIVRO;

CREATE GENERATOR GEN_IDLIVRO_ITEM;

CREATE GENERATOR GEN_IDPESSOA;

CREATE GENERATOR GEN_IDEMPRESTIMO;

CREATE GENERATOR GEN_IDEMPRESTIMO_ITEM;

CREATE TABLE EDITORA
(
  IDEDITORA INTEGER NOT NULL,
  EDITORA VARCHAR(100) NOT NULL
);

CREATE TABLE LIVRO
(
  IDLIVRO INTEGER NOT NULL,
  IDEDITORA INTEGER NOT NULL,
  LIVRO VARCHAR(100) NOT NULL,
  QUANTIDADE INTEGER NOT NULL
);

CREATE TABLE LIVRO_ITEM
(
  IDLIVRO_ITEM INTEGER NOT NULL,
  IDLIVRO INTEGER NOT NULL,
  ITEM INTEGER NOT NULL,
  STATUS CHAR(1) NOT NULL,
  DATA_ENTRADA DATE
);

CREATE TABLE PESSOA
(
  IDPESSOA INTEGER NOT NULL,
  PESSOA VARCHAR(100) NOT NULL,
  ENDERECO VARCHAR(100) NOT NULL,
  BAIRRO VARCHAR(100) NOT NULL,
  CIDADE VARCHAR(100) NOT NULL,
  ESTADO CHAR(2) NOT NULL,
  TELEFONE VARCHAR(11) NOT NULL
);

CREATE TABLE EMPRESTIMO
(
  IDEMPRESTIMO INTEGER NOT NULL,
  IDPESSOA INTEGER NOT NULL,
  DATA_EMPRESTIMO DATE NOT NULL
);

CREATE TABLE EMPRESTIMO_ITEM
(
  IDEMPRESTIMO_ITEM INTEGER NOT NULL,
  IDEMPRESTIMO INTEGER NOT NULL,
  IDLIVRO_ITEM INTEGER NOT NULL,
  DATA_PREVISTA DATE NOT NULL,
  DATA_DEVOLUCAO DATE
);

ALTER TABLE EDITORA ADD CONSTRAINT PK_IDEDITORA
PRIMARY KEY(IDEDITORA) USING INDEX IDX_EDITORA;

ALTER TABLE LIVRO ADD CONSTRAINT PK_IDLIVRO
PRIMARY KEY(IDLIVRO) USING INDEX IDX_IDLIVRO;

ALTER TABLE LIVRO ADD CONSTRAINT FK_LIVRO_EDITORA
FOREIGN KEY(IDLIVRO) REFERENCES EDITORA
USING INDEX IDX_LIVRO_EDITORA;

ALTER TABLE LIVRO_ITEM ADD CONSTRAINT PK_IDLIVRO_ITEM
PRIMARY KEY(IDLIVRO_ITEM) USING INDEX IDX_IDLIVRO_ITEM;

ALTER TABLE LIVRO ADD CONSTRAINT FK_LIVRO_EDITORA
FOREIGN KEY(IDEDITORA) REFERENCES EDITORA
USING INDEX IDX_LIVRO_EDITORA;

ALTER TABLE PESSOA ADD CONSTRAINT PK_IDPESSOA
PRIMARY KEY(IDPESSOA) USING INDEX IDX_IDPESSOA;

ALTER TABLE EMPRESTIMO ADD CONSTRAINT PK_IDEMPRESTIMO
PRIMARY KEY(IDEMPRESTIMO) USING INDEX IDX_IDEMPRESTIMO;

ALTER TABLE EMPRESTIMO ADD CONSTRAINT FK_EMPRESTIMO_PESSOA
FOREIGN KEY(IDPESSOA) REFERENCES PESSOA
USING INDEX IDX_EMPRESTIMO_PESSOA;

ALTER TABLE EMPRESTIMO_ITEM ADD CONSTRAINT FK_EMPRESTIMO_EMP_ITEM
FOREIGN KEY(IDEMPRESTIMO) REFERENCES EMPRESTIMO
ON DELETE CASCADE
ON UPDATE CASCADE
USING INDEX IDX_EMPRESTIMO_EMP_ITEM;

ALTER TABLE EMPRESTIMO_ITEM ADD CONSTRAINT PK_EMPRESTIMO_ITEM
PRIMARY KEY(IDEMPRESTIMO_ITEM) USING INDEX IDX_EMPRESTIMO_ITEM;

ALTER TABLE EMPRESTIMO_ITEM ADD CONSTRAINT FK_EMPRESTIMO_LIVRO_
ITEM
FOREIGN KEY(IDLIVRO_ITEM) REFERENCES LIVRO_ITEM
USING INDEX IDX_EMPRESTIMO_LIVRO_ITEM;
```

Criando o servidor de aplicação

Agora que concluímos a criação do nosso banco de dados, vamos dar início a implementação do servidor de aplicação, durante este processo irei passar alguns detalhes importantes para a configuração do nosso ambiente de desenvolvimento.

Abra seu *RAD Studio 2007* e vá em *File>New>VCL Forms Application – Delphi for win 32*, após isso modifique no seu *Form* as propriedades *Name* para *frmServidor* e *Caption* para *Servidor de aplicação BSS*, adicione ao *form* um componente *Label* e altere sua propriedade *Caption* para *Servidor de aplicação BSS*, por último adicione um componente *SQLConnection* e altere as propriedades *Name* para *sqlConexao*, *LoginPrompt* para *False*. Em seguida, configure sua conexão com o banco de dados conforme mostra o exemplo da **Figura 3** e altere a propriedade *VendorLib* do seu *SQLConnection* para *fbclient.dll*, ao terminar seu *Form deverá* estar com a aparência semelhante a **Figura 4**.

Feito isso, adicione à cláusula *uses* do seu formulário as *units Provider* e *DBClient* para que possamos utilizar componentes dos tipos *ClientDataSet e DataSetProvider* em tempo de execução, após a declaração, vamos criar alguns métodos genéricos que poderão ser reutilizados em nossa aplicação servidora a qualquer momento.

Primeiramente declararemos a função *ExecuteSQL* que será responsável por receber uma instrução *SQL* e retornar o *Data Package* em forma de um *OleVariant* e a função *GetId* que será responsável por receber como parâmetro o nome do campo chave primária e resultar o valor correspondente ao *Generator* no banco de dados. Declare na seção *public* do seu *Form* o cabeçalho das rotinas conforme mostra a **Listagem 2**, e pressione simultaneamente as teclas *Ctrl + Shift + C* para que os corpos dos métodos sejam criados automaticamente, em seguida faça a implementação conforme mostrado na **Listagem 3**.

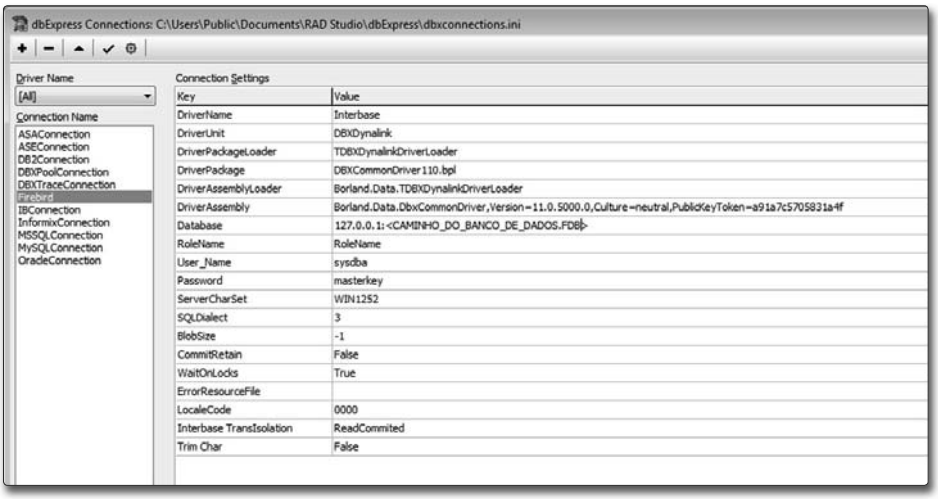


Figura 3. Formulário de conexão com o banco de dados

Figura 4. Formulário do servidor de Aplicação

Nota

Um OleVariant é um tipo de variável muito utilizado para o transporte de dados em arquitetura multicamadas, é responsável por trafegar o data package através de um array de dados de tipo Variant.

Não há nenhum segredo na função, vejamos. Nas primeiras linhas declaramos as variáveis necessárias para a criação das conexões com o banco de dados dinamicamente. Perceba que criamos um *SQLDataSet*, um *DataSetProvider* e um *ClientDataSet*. Em seguida *startamos* uma transação usando o método *StartTransation* e então atribuímos a variável *SQL* (que veio por parâmetro) na *procedure*. Resultamos o *Data* do *ClientDataSet* auxiliar, que na verdade são os dados propriamente ditos. O método *GetID* apenas executa uma instrução *SQL* no banco de dados para então gerar a nova chave para esse registro.

Em seguida salve o seu projeto utilizando o menu *File>Save All*, nomeando a *unit* como *untServidor* e o projeto como *ServerBSS.dpr*.

Finalmente iremos dar início a parte mais interessante da criação do servidor de aplicação, acesse o menu *File>New>Other>Multtier>Remote Data Module*. Após a seleção, irá ser exibida uma janela de *Wizard* conforme mostra a **Figura 5**, nomeie a *CoClass Name* para *RDMPrincipal*, e selecione respectivamente *Multiple Instance* e *Single* para as caixas de seleção *Instancing* e *Threading Model*.

Listagem 2. Cabeçalho dos Métodos do servidor

```
unit untServidor;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
   Controls, Forms,
  Dialogs, StdCtrls, WideStrings, DB, SqlExpr, DBClient,
   Provider;

type
  TfrmServidor = class(TForm)
    Label1: TLabel;
    sqlConexao: TSQLConnection;
  private
    { Private declarations }
  public
    function GetId(KeyName : String) : Integer;
    function ExecuteSQL(SQL : String) : OleVariant;
    { Public declarations }
  end;

var
  frmServidor: TfrmServidor;

implementation

{$R *.dfm}
```

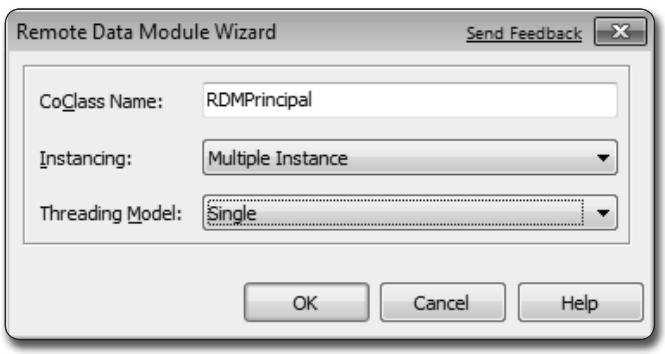


Figura 5. Janela assistente para a criação do Remote Data Module

Listagem 3. Implementação das rotinas do servidor

```
function TfrmServidor.ExecuteSQL(SQL: String): OleVariant;
var
  Transacao   : TTransactionDesc;
  sqlAuxiliar : TSQLDataSet;
  dspAuxiliar : TDataSetProvider;
  cdsAuxiliar : TClientDataSet;
begin
  sqlAuxiliar := TSQLDataSet.Create(nil);
  sqlAuxiliar.SQLConnection := sqlConexao;
  dspAuxiliar := TDataSetProvider.Create(nil);
  dspAuxiliar.DataSet := sqlAuxiliar;

  cdsAuxiliar := TClientDataSet.Create(nil);
  cdsAuxiliar.SetProvider(dspAuxiliar);
  try
    try
      Transacao.TransactionID := 1;
      Transacao.IsolationLevel := xilREADCOMMITTED;
      sqlConexao.StartTransaction(Transacao);
      sqlAuxiliar.CommandText := SQL;
      cdsAuxiliar.Open;
      Result := cdsAuxiliar.Data;
      sqlConexao.Commit(Transacao);
    except
      if sqlConexao.InTransaction then
        sqlConexao.Rollback(Transacao);
      Result := Null;
    end;
  finally
    cdsAuxiliar.Close;
    FreeAndNil(cdsAuxiliar);
    FreeAndNil(dspAuxiliar);
    FreeAndNil(sqlAuxiliar);
  end;
end;

function TfrmServidor.GetId(KeyName: String): Integer;
var
  cds : TClientDataSet;
begin
  try
    try
      cds := TClientDataSet.Create(nil);
      cds.Data := ExecuteSQL('SELECT GEN_ID(GEN_' + KeyName + ',1)'
       + ' FROM RDB$DATABASE ');
      Result := cds.Fields[0].Value;
    except
      Result := -1;
    end;
  finally
    FreeAndNil(cds);
  end;
end;
```

Após criar o *Remote Data Module* adicione à clausula *uses* a *unit untServidor,* e adicione quatro componentes *SQLDataSet* apontando suas propriedades *SQLConnection* ao componente *SQLConexao.* Altere a propriedade *Name* para *sqlEditora*, *sqlPessoa*, *sqlLivro* e *sqlLivro_Item* e adicione os comandos *SQL* da **Listagem 4** à propriedade *CommandText*.

Tendo adicionado as instruções *SQL,* é hora de configurarmos os parâmetros de cada um dos nossos componentes *SQLDataSet*. Selecione-os ignorando o componente *sqlLivro_Item* e

Nota do DevMan

Os Remote Data Modules são fundamentais em aplicações multicamadas e são utilizados quando a arquitetura utiliza comunicação por meio de DCOM, Sockets ou HTTP, estes conseguem prover a comunicação entre as camadas cliente e servidora devido ao fato de implementarem a interface IAppServer.

Listagem 4. Comandos SQL dos Componentes do RDMPrincipal

```
[sqlEditora]
SELECT
  EDITORA.IDEDITORA,
  EDITORA.EDITORA
FROM
  EDITORA
WHERE
  EDITORA.IDEDITORA = :IDEDITORA

[sqlPessoa]
SELECT
  PESSOA.IDPESSOA,
  PESSOA.PESSOA,
  PESSOA.ENDERECO,
  PESSOA.BAIRRO,
  PESSOA.CIDADE,
  PESSOA.ESTADO,
  PESSOA.TELEFONE
FROM
  PESSOA
WHERE
  PESSOA.IDPESSOA = :IDPESSOA

[sqlLivro]
SELECT
  LIVRO.IDLIVRO,
  LIVRO.IDEDITORA,
  LIVRO.LIVRO,
  LIVRO.QUANTIDADE,
  EDITORA.EDITORA
FROM
  LIVRO
LEFT JOIN EDITORA ON
  LIVRO.IDEDITORA = EDITORA.IDEDITORA
WHERE
  LIVRO.IDLIVRO = :IDLIVRO

[sqlLivro_Item]
SELECT
  LIVRO_ITEM.IDLIVRO_ITEM,
  LIVRO_ITEM.IDLIVRO,
  LIVRO_ITEM.ITEM,
  LIVRO_ITEM.STATUS,
  LIVRO_ITEM.DATA_ENTRADA
FROM
  LIVRO_ITEM
WHERE
  LIVRO_ITEM.IDLIVRO = :IDLIVRO
```

acione a propriedade *Params*, feito isso, será exibido o editor de parâmetros. Configure em cada um dos parâmetros a propriedade *DataType* para *ftInteger.* Agora adicione três componentes *DataSetProvider* e um componente *DataSource,* alterando as propriedades *Name* dos *DataSetProvider* para *dspEditora*, *dspPessoa* e *dspLivro* e a mesma propriedade para o componente *DataSource* renomeando-o para *dsrMestreDetalhe.* Selecione cada um deles e configure a propriedade *UpdateMode* para *upWhereKeyOnly,* ainda nos componentes *DataSetProvider* dê um clique na propriedade *Options* e altere as opções *poIncFieldProps* e *poPropagateChanges* para *True,* isto fará com que todas as configurações como máscaras e *ProviderFlags* feitas nos *Fields* dos *sqlDataSet* sejam repassadas automaticamente para os *Fields* dos componentes *ClientDataSet*. Marque também em *Options* apenas no componente *dspLivro* as opções *poCascadeDeletes* e *poCascadeUpdates* para *True*, essas opções marcadas fazem com que ao modificar ou deletar um livro todos os itens referentes recebam as alterações, ou em caso de deleção, estes serão todos deletados. Agora basta selecionarmos o componente *DataSource* e apontarmos a propriedade *DataSet* para o *sqlLivro,* selecione o *sqlLivro_Item* e vincule sua propriedade *DataSource* ao *dsrMestreDetalhe,* e finalmente acesse a propriedade *Params* configurando o parâmetro para *ftInteger.*

Listagem 5. Códigos dos Eventos BeforeUpdateRecord dos DataSetProvider

```
[dspEditora]
procedure TRDMPrincipal.dspEditoraBeforeUpdateRecord(Sender:
 TObject;
  SourceDS: TDataSet; DeltaDS: TCustomClientDataSet;
   UpdateKind:TUpdateKind;
  var Applied: Boolean);
begin
  if UpdateKind = ukInsert then
    DeltaDS.FieldByName('IDEDITORA').NewValue :=
      frmServidor.GetId('IDEDITORA');
end;

[dspPessoa]
procedure TRDMPrincipal.dspPessoaBeforeUpdateRecord
 (Sender:TObject;
  SourceDS: TDataSet; DeltaDS: TCustomClientDataSet;
   UpdateKind:TUpdateKind;
  var Applied: Boolean);
begin
  if UpdateKind = ukInsert then
    DeltaDs.FieldByName('IDPESSOA').NewValue :=
     frmServidor.GetId('IDPESSOA');
end;

[dspLivro]
procedure TRDMPrincipal.dspLivroBeforeUpdateRecord(
 Sender:TObject;
  SourceDS: TDataSet; DeltaDS: TCustomClientDataSet;
   UpdateKind:TUpdateKind;
  var Applied: Boolean);
{$J+}
const
  inChave : Integer = -1;
{$J-}
begin
  if UpdateKind = ukInsert then
    begin
      if SourceDS = sqlLivro then
        begin
          inChave := frmServidor.GetId('IDLIVRO');
          DeltaDS.FieldByName('IDLIVRO').NewValue :=
           inChave;
        end
      else if SourceDS = sqlLivro_Item then
        begin
          if DeltaDS.FieldByName('IDLIVRO').Value =
           0 then
            DeltaDS.FieldByName('IDLIVRO').NewValue :=
              inChave;
          DeltaDS.FieldByName('IDLIVRO_ITEM').
           NewValue :=
            frmServidor.GetId('IDLIVRO_ITEM');
        end;
    end;
end;
```

Após realizar estas modificações, todas as ligações dos nossos componentes base estarão prontas, e seu *RemoteDataModule* deve estar parecido com o exemplo da **Figura 6,** porém agora ainda é necessário configurar algumas propriedades extras dos *Fields* de cada um dos nossos *SQLDataSet*. Selecione seu componente *sqlEditora*¸ clique com o botão direito do mouse e selecione a opção *FieldsEditor*, clique com o botão direito do mouse no *FieldsEditor* e selecione então *AddAllFields*, tendo adicionado os *Fields* é hora de partirmos para os *ProviderFlags*, lembrando que a configuração deles é extremamente importante para o modo de atualização e boa performance gerenciada pelo *DataSetProvider*.

A configuração deve ser feita conforme o tipo do campo, sendo que para os campos chave os *ProviderFlags* devem estar marcados como *True* nas opções [*pfInUpdate*, *pfInKey*, *pfInWhere*], para os campos da própria tabela que serão atualizados apenas marque como [*pfInUpdate*] nos demais campos sempre deixe todos desmarcados []. No caso do *sqlEditora* configure o campo chave (*IDEDITORA*) para [*pfInUpdate*, *pfInKey*, *pfInWhere*] e altere a propriedade *DisplayFormat* para *000000* e no campo *EDITORA* altere os *ProviderFlags* para [*pfInUpdate*], repita a configuração para os demais campos e informe este modelo de *DisplayFormat* para todos os campos chave e para o *field ITEM* localizado no *SQLDataSet sqlLivro_Item*.

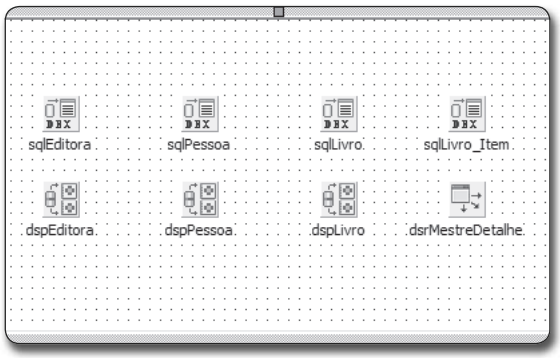
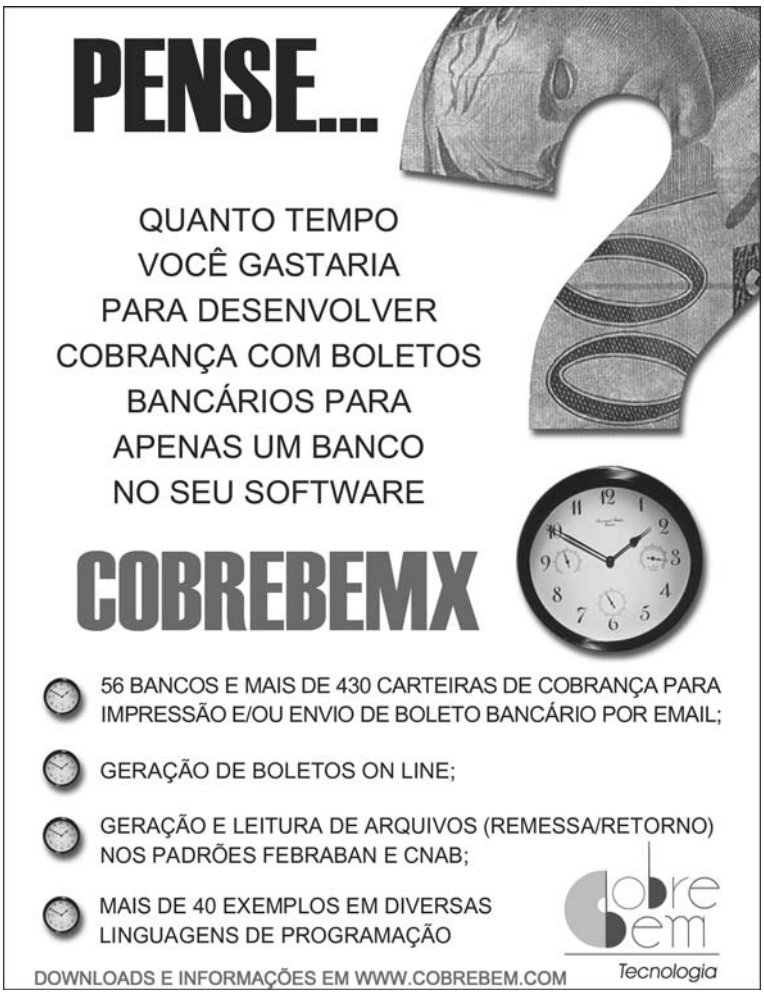


Figura 6. Exemplo do RemoteDataModule Principal

Agora que ajustamos os pequenos detalhes, é hora de prepararmos os nossos *DataSetProvider* para que quando os dados forem enviados ao servidor para serem salvos, os próprios componentes se encarreguem



de obter a chave primária real do campo. Para isto, vamos utilizar o evento *BeforeUpdateRecord* do *DataSetProvider*, codifique os eventos referentes a cada um dos componentes conforme mostra a **Listagem 5.**

O evento *BeforeUpdateRecord* ocorre conforme a quantidade de registros existentes no *DataPackage*, ou seja, caso tenhamos dez registros, por exemplo, este evento ocorrerá dez vezes.

Observando os códigos da **Listagem 5** repare que o evento *BeforeUpdateRecord* possui quatro parâmetros e uma variável. O parâmetro *Sender* é o próprio objeto, *SourceDs* indica qual *DataSet* está passando por aquele evento, *DeltaDs* recebe o objeto *ClientDataSet* que enviou o *DataPackage*,e *UpdateKind* informa qual é a ação que o *DataSetProvider* deve tomar podendo ser *ukDelete*, *ukInsert* ou *ukModified*, repare que os eventos dos *DataSetProvider dspPessoa* e *dspEditora* são semelhantes, apenas verificam se o estado atual é de inserção e caso a condição seja satisfeita, o campo chave do *DataPackage* recebe o novo valor através da função criada no *frmServidor* que retorna o número do *Generator* do banco de dados. A codificação do *dspLivro* é um pouco mais interessante, repare que temos as diretivas *{$J+}* e *{$J-,}*. Através delas criamos um tipo de constante deferente das comuns, esta nos permite que possamos atribuir valores. Desse modo, ao receber o valor da chave, logo que o evento for passado novamente, a constante manterá o valor atribuído e podendo ser reaproveitado, com isso, verificamos se o *UpdateKind* é de inserção, pois, satisfazendo esta validação, os *DataSet's* necessitarão de receber o valor correto, devido ao fato deste relacionamento ser um mestre-detalhe. Os dois *DataSet* que estão ligados passarão por este evento, então é testado qual é o *DataSet* através do parâmetro *SourceDS* (lembrando que sempre é o Mestre o primeiro a salvar os dados). Nesse momento, atribuímos o valor da chave do Mestre para repassarmos para o *Detalhe*, que recebe a chave do *Mestre* e o valor de sua própria chave.

Realizadas todas as codificações, agora vamos criar outro *RemoteDataModule* que será responsável pelos empréstimos de livros

Listagem 6. SQL relativos ao Empréstimo

```
[sqlEmprestimo]
SELECT
  EMPRESTIMO.IDEMPRESTIMO,
  EMPRESTIMO.IDPESSOA,

  EMPRESTIMO.DATA_EMPRESTIMO
FROM
  EMPRESTIMO
WHERE
  EMPRESTIMO.IDEMPRESTIMO =:IDEMPRESTIMO

[sqlEmprestimo_Item]
SELECT
  EMPRESTIMO_ITEM.IDEMPRESTIMO_ITEM,
  EMPRESTIMO_ITEM.IDLIVRO_ITEM,
  EMPRESTIMO_ITEM.DATA_PREVISTA,
  EMPRESTIMO_ITEM.DATA_DEVOLUCAO,
  LIVRO.LIVRO
FROM
  EMPRESTIMO_ITEM
LEFT JOIN LIVRO_ITEM ON
  EMPRESTIMO_ITEM.IDLIVRO_ITEM = LIVRO_ITEM.IDLIVRO_ITEM
LEFT JOIN LIVRO ON
  LIVRO_ITEM.IDLIVRO = LIVRO.IDLIVRO
WHERE
  EMPRESTIMO_ITEM.IDEMPRESTIMO =:IDEMPRESTIMO
```

Listagem 7. Evento BeforeUpdateRecord do DataSetProvider dspEmprestimo

```
[dspEmprestimo]
procedure TRDMEmprestimo.dspEmprestimoBeforeUpdateRecord
 (Sende r: TObject;
  SourceDS: TDataSet; DeltaDS: TCustomClientDataSet; UpdateKind:
   TUpdateKind;
  var Applied: Boolean);
{$J+}
  const
    Key : Integer = -1;
{$J-}
begin
  if UpdateKind = ukInsert then
    begin
      if SourceDS = sqlEmprestimo then
        begin
          Key := frmServidor.GetId('IDEMPRESTIMO');
          DeltaDS.FieldByName('IDEMPRESTIMO').NewValue := Key;
        end
      else if SourceDS = sqlEmprestimo_Item then
        begin
          if DeltaDS.FieldByName('IDEMPRESTIMO').Value = 0 then
            DeltaDS.FieldByName('IDEMPRESTIMO').NewValue := Key;
          DeltaDS.FieldByName('IDEMPRESTIMO_ITEM').NewValue :=
            frmServidor.GetId('IDEMPRESTIMO_ITEM');
        end;
    end;
end;
```

Listagem 8. Criação da variável de instância do objeto COM

```
unit untRDMEmprestimo;

{$WARN SYMBOL_PLATFORM OFF}

interface

uses
  Windows, Messages, SysUtils, Classes, ComServ, ComObj, VCLCom,
   DataBkr,
  DBClient, ServerBSS_TLB, StdVcl, FMTBcd, DB, Provider,
   SqlExpr;

type
  TRDMEmprestimo = class(TRemoteDataModule, IRDMEmprestimo)
    sqlEmprestimo: TSQLDataSet;
    sqlEmprestimo_Item: TSQLDataSet;
    dspEmprestimo: TDataSetProvider;
    dsrEmprestimo: TDataSource;
    sqlEmprestimoIDEMPRESTIMO: TIntegerField;
    sqlEmprestimoIDPESSOA: TIntegerField;
    sqlEmprestimoDATA_EMPRESTIMO: TDateField;
    sqlEmprestimo_ItemIDEMPRESTIMO_ITEM: TIntegerField;
    sqlEmprestimo_ItemIDLIVRO_ITEM: TIntegerField;
    sqlEmprestimo_ItemDATA_PREVISTA: TDateField;
    sqlEmprestimo_ItemDATA_DEVOLUCAO: TDateField;
    sqlEmprestimo_ItemLIVRO: TStringField;
    procedure dspEmprestimoBeforeUpdateRecord(Sender: TObject;
      SourceDS: TDataSet; DeltaDS: TCustomClientDataSet;
      UpdateKind: TUpdateKind; var Applied: Boolean);
  private
    { Private declarations }
  protected
    class procedure UpdateRegistry(Register: Boolean; const
     ClassID, ProgID: string); override;
  public
    { Public declarations }
  end;

var
  EmprestimoComponentFactory : TComponentFactory;

implementation

uses untServidor;
```

Listagem 9. Criando a instância do RDMFilho para o RDMPai

```
initialization
  EmprestimoComponentFactory :=
    TComponentFactory.Create(ComServer, TRDMEmprestimo,
    Class_RDMEmprestimo, ciMultiInstance, tmSingle);
```

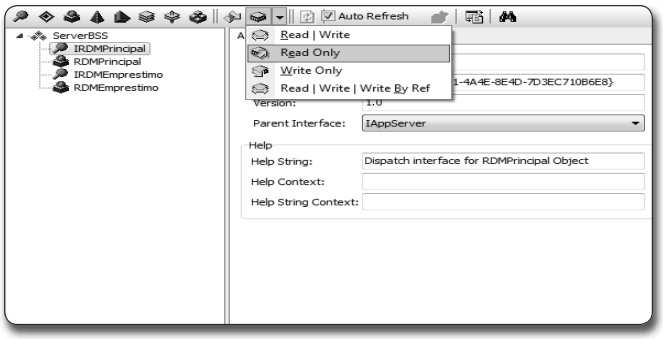


Figura 7. Criando métodos via Type Library

e que será instanciado para o *RDMPrincipal*, fazendo com que o novo objeto *COM* seja visível na camada *Cliente*. Mãos a obra então, vá em *File>New>Other* e selecione novamente a categoria *Multitier>RemoteDataModule*. Na nova caixa de *Wizard* nomeie o *CoClassName* para *RDMEmprestimo, Instancing* para *Multiple Instance* e *Threading Model* para *Single* e clique em *Ok*. Adicione em sua cláusula *uses* o *untServidor* e salve-o em seguida com o nome de *untRDMEmprestimo*, agora insira dois *SQLDataSet* e altere suas propriedades *Name* para *sqlEmprestimo* e *sqlEmprestimo_Item* vincule o componente *sqlConexao* do *frmServidor* à propriedade *SQLConnection* e configure nas propriedades *CommandText* de cada um, as instruções *SQL* de acordo com a **Listagem 6.** Adicione os parâmetros nas propriedades *Params*, definido-os como *ftInteger*, depois adicione seus *Fields* no *FieldsEditor* e configure os *ProviderFlags*,do mesmo modo em que foi feito nos anteriores, agora adicione um *DataSetProvider* e renomeie-o para *dspEmprestimo*, altere a propriedade *UpdateMode* para *upWhereKeyOnly* e em *Options* habilite [*poCascadeDeletes*, *poCascadeUpdates*, *poIncFieldProps*, *poPropagateChanges*] agora ligue a propriedade *DataSet* ao *sqlEmprestimo*, por último adicione um *DataSource* e altere a propriedade *Name* para *dsrEmprestimo*, vinculando sua propriedade *DataSet* ao *sqlEmprestimo*,e setando este *DataSource* à propriedade *DataSource* do nosso *sqlEmprestimo_Item*, finalizando o processo teremos mais um relacionamento *Mestre-detalhe*.

Vamos novamente configurar o evento BeforeUpdateRecord do componente dspEmprestimo do mesmo modo em que configuramos para o dspMestreDetalhe no RDMPrincipal veja a codificação do evento pronta conforme mostra a **Listagem 7.**

Após tudo configurado é hora de fazermos o *RDMPrincipal enxergar* o *RDMEmprestimo*, para que a nossa camada Cliente possa visualizar as instâncias secundárias (*Filhos*) que são acessadas através do *RDMPrincipal*. Para isso, vamos criar uma variável global do tipo *TComponentFactory* conforme mostra a **Listagem 8.** Observando o rodapé do *RDMEmprestimo* podemos perceber a declaração abaixo da cláusula *initialization* que mostra a criação do tipo de objeto *COM*, utilizaremos esta variável global que foi declarada para instanciar este objeto ao *RDMPrincipal* como mostra a **Listagem 9.**

O próximo passo é criar a função que retornará o objeto em si, utilizaremos a *Type Library* que funciona como uma espécie de *Wizard* para a criação dos métodos no servidor de aplicação. Acesse o menu *View>Type Library* e selecione do lado esquerdo o *IRDMPrincipal (Interface)*, clique no botão *New Method* conforme indica a **Figura 7** selecionando um método do tipo *Read Only*. Após isso, do lado direito você terá três informações *Name*, *ID* e *Type*, para *Name* informe *RDMEmprestimo*, o *ID* não é necessário ser alterado e em *Type* vamos indicar a *Interface* do próprio objeto que queremos retornar no caso o objeto *RDMEmprestimo*, então procure por *IRDMEmprestimo*. Salve suas alterações e abra o *RDMPrincipal*, veja que foi criada uma função chamada *Get_RDMEmprestimo*, porém sem implementação alguma, adicione na cláusula *uses* do seu *RDMPrincipal* o *untRDMEmprestimo* e faça a implementação conforme o código a seguir**.** Através desta codificação nosso servidor de aplicação já consegue retornar o objeto filho para a visualização na camada *Cliente*. Após termos feito isso o nosso modelo de *Servidor* está pronto para rodar a aplicação, agora daremos início ao desenvolvimento da camada *Cliente*.

```
function TRDMPrincipal.Get_RDMEmprestimo: IRDMEmprestimo;
begin
  Result := EmprestimoComponentFactory.CreateComObject(nil) as
    IRDMEmprestimo;
end;
```

Conclusão

Como vimos, o conceito de multicamadas é uma ótima maneira de utilizarmos sistemas *win32* na *web*. As divisões das camadas tornam a manutenção do desenvolvimento, e a distribuição das aplicações de grande disponibilidade muito mais fáceis e ágeis. Uma boa opção é ler o próprio *help* do *Delphi* para obter informações mais claras sobre os tipos de instância e os modelos de threads, utilizando-os conforme gosto e escopo da aplicação.

Até a próxima e até mais. ●

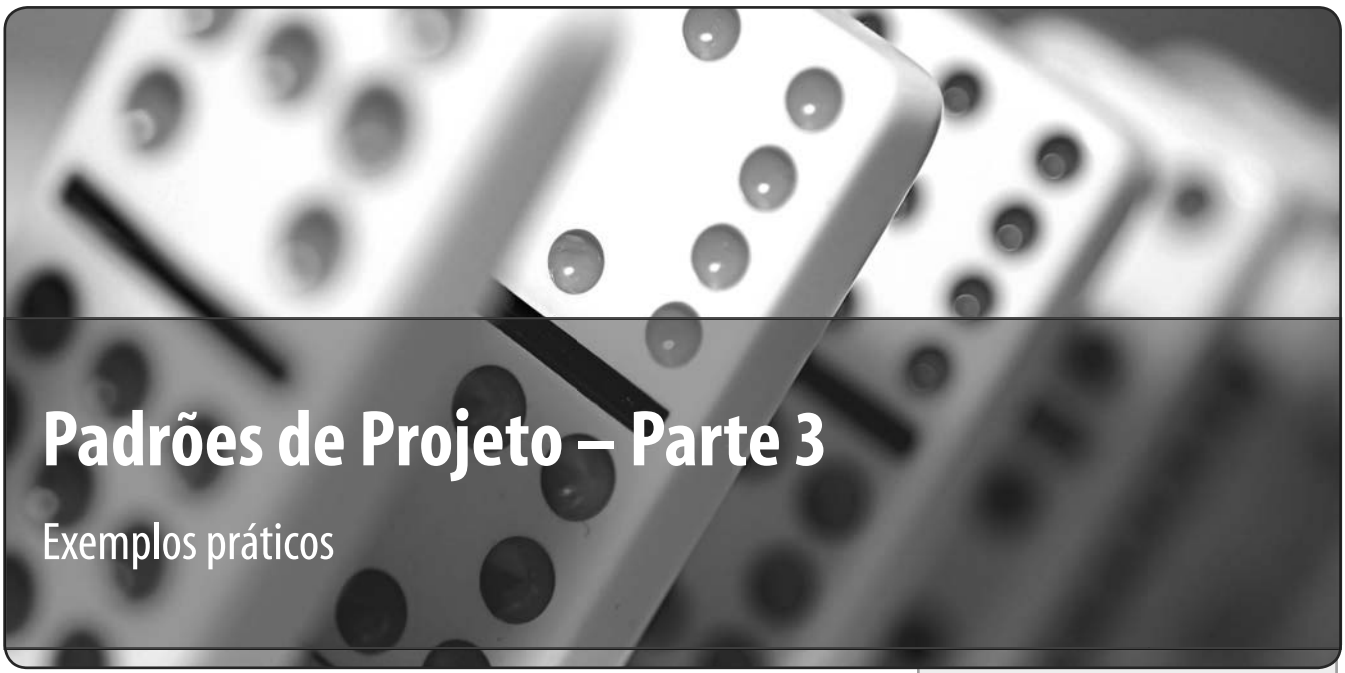
Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos intermediários sobre Delphi Win 32 e Delphi for .NET



Padrões de Projeto – Parte 3

Exemplos práticos

Nesse artigo veremos

- Quais são os padrões comportamentais;
- Como implementá-los em Delphi;
- Como utilizá-los em nosso dia a dia através de exemplos práticos.

Qual a finalidade

Mostrar o uso adequado de padrões de projeto em uma aplicação Delphi.

Quais situações utilizam esse recurso?

No desenvolvimento de sistemas em geral.

Paulo Quicoli
pauloquicoli@devmedia.com.br
Editor técnico da revista ClubeDelphi, formado em processamento de dados pela FATEC-TQ. Atua como analista e desenvolvedor da ControlM Informática (www.controlm.com.br). Utiliza Delphi desde sua primeira versão, entusiasta do desenvolvimento orientado a objetos tem publicado vários artigos sobre o assunto. Blog: www.pauloquicoli.spaces.live.com .

Resumo do DevMan

Os padrões de projeto são soluções para problemas recorrentes que surgem no desenvolvimento de sistemas. Foram inicialmente catalogados pela GoF (Gang of Four). Quatro autores, Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides, que lançaram o livro mais famoso sobre o assunto: Design Patterns: Elements of Reusable Obejct-Oriented Software. Considerados como boa prática entre os desenvolvedores, veremos como aplicar os padrões comportamentais em Delphi, em situações reais que podem acontecer no dia a dia do trabalho.

É certo que o assunto *Padrões de Projeto* é extremamente grande e que há uma vasta documentação disponível na internet, em livros, em acervos digitais e etc. Infelizmente falar de todos faz com que acabemos repetindo conteúdos e conceitos, por isso essa é nossa terceira e última parte sobre padrões de projetos e veremos os padrões comportamentais. Estes permitem que possamos ajustar o comportamento de objetos de forma flexível. Os seguintes padrões formam essa categoria.

- *Chain of responsability;*
- *Command;*
- *Interpreter;*
- *Iterator;*
- *Mediator;*
- *Memento;*
- *Observer;*
- *State;*
- *Strategy;*
- *Template method;*
- *Visitor.*

Padrão Command

Posso dizer que o padrão *Command* é o mais utilizado entre a comunidade *Delphi*, mesmo que *sem saber*. Esse padrão está implementado como um recurso da *VCL* do *Delphi*. Vamos entender primeiro para que esse padrão é utilizado.

Ele encapsula uma ação ou solicitação como um objeto. É comumente utilizado em menus onde cada opção representa uma *ação*. Todas as ações são criadas em objetos que possuem a mesma *interface*, de tal forma que possam ser manipulados de forma polimórfica. Isso significa que, por terem a mesma *assinatura* é possível substituir um comando por outro. Tudo isso nos remete ao objeto *TAction* e *TActionList* que permitem centralizar as ações de um menu ou outro objeto que faça uso de *actions*, como o *TButton* por exemplo.

Implementação

Vamos fazer um pequeno exemplo do uso de objetos *TAction*, contudo sem utilizar um *TActionList*. Quero mostrar como programar as ações por código. Muitas vezes estamos tão acostumados a utilizar os assistentes que o *Delphi* nos oferece que esquecemos que podemos escrever código para alterar algum comportamento padrão. Crie um novo projeto, do tipo *Win32* e no formulário principal adicione um *TButton*, *TRadioGroup* e um *TEdit*, deixando-os como na **Figura 1**.

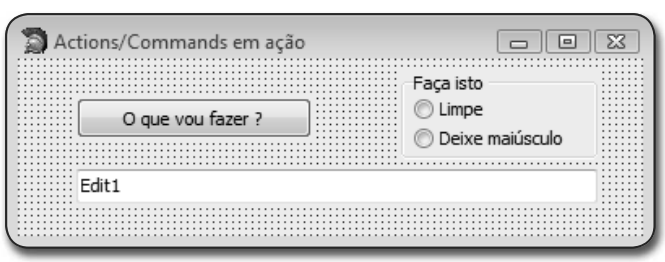


Figura 1. Formulário principal

A idéia deste exemplo é trocar, em tempo de execução, a *TAction* que estará associada o botão, permitindo que ele mude seu *comportamento* de acordo com o que está selecionado no *RadioGroup*, ou seja, ele poderá limpar o conteúdo do *Edit* ou deixar seu texto em maiúsculo. A primeira coisa a se fazer então é criar dois objetos do tipo *TAction*. Na seção *private* do formulário os crie conforme código a seguir.

```
private
 { Private declarations }
 actLimparEdit: TAction;
 actMudarCaixa: TAction;
```

Não podemos nos esquecer de adicionar ao *Uses* uma referência à *Unit AcntList*, pois é nela que a classe *TAction* está definida. A seguir vamos criar dois métodos que serão responsáveis pelo que o as ações executarão. Como é sabido, quando um botão possui um *TAction* associado a ele, seu evento *onClick* dispara o evento *onExecute* do *TAction*. Vamos criar dois métodos na seção *private*, cada um para sua *action*, que serão disparados pelo evento *onExecute* citado. Crie conforme **Listagem 1**.

Listagem 1. Métodos que serão executados

```
01 procedure TForm1.Limpar(Sender: TObject);
02 begin
03   Edit1.Clear;
04 end;
05
06 procedure TForm1.MudarCaixa(Sender: TObject);
07 begin
08   Edit1.Text := UpperCase(Edit1.Text);
09 end;
```

O código dos dois métodos é tão simples que dispensa comentários. O próximo passo é ligar esses métodos ao evento *onExecute* das *actions*. Vamos criar outro método chamado *InicializaActions* e implementá-lo como na **Listagem 2**.

Listagem 2. Métodos InicializaActions

```
01 procedure TForm1.inicializaActions;
02 begin
03   actLimparEdit := TAction.Create(nil);
04   actLimparEdit.Caption := 'Limpar';
05   actLimparEdit.OnExecute := Limpar;
06
07   actMudarCaixa := TAction.Create(nil);
08   actMudarCaixa.Caption := 'Deixar maiúsculo';
09   actMudarCaixa.OnExecute := MudarCaixa;
10 end;
```

Na linha 3 e 7 instanciamos os objetos *TAction*. Então definimos sua propriedade *Caption*, que será exibida no botão, na linha 4 e 8. Já na linha 5 e 9 estamos ligando o evento *OnExecute* aos métodos anteriormente criados. Como o método *inicializaActions* criado é preciso chamá-lo no evento *OnCreate* do formulário para que tudo esteja pronto. Agora vamos ligar ao botão a *action* de acordo com a escolha do usuário no *RadioGroup*. Isso pode ser visto na **Listagem 3**.

Listagem 3. Escolhendo a ação a ser feita

```
01 procedure TForm1.RadioGroup1Click(Sender: TObject);
02 begin
03   case RadioGroup1.ItemIndex of
04     0 : Button1.Action := actLimparEdit;
05     1 : Button1.Action := actMudarCaixa;
06   end;
07 end;
```

Observe as linhas 4 e 5. Nelas estamos ligando as instâncias, objetos, de *TAction* à propriedade *Action* do botão. Dessa forma o botão pode executar operações diferentes

mediante alguma parametrização, que em nosso caso é feita pelo *RadioGroup*. Execute a aplicação e veja o botão se comportando diferentemente.

Padrão Interpreter

Este padrão é utilizado quando se necessita interpretar uma determinada gramática. Alguns frameworks de impressão como o *FastReport* possuem essa implementação. Esse recurso nele é implementado por permitir digitar código *Delphi* dentro do próprio relatório e esse por sua vez é interpretado e executado durante a geração do relatório. Tudo isso se resume a ferramenta de *script*. Caso seu sistema venha necessitar algo assim a melhor solução é procurar por componentes de terceiros que já possuem o recurso do que tentar criar o seu próprio. Porém existe outra situação muito corriqueira onde empregamos o padrão *interpreter*. Quando vamos por exemplo converter um valor número pela sua representação por extenso. Pegamos uma gramática numérica, 1,2,3, etc e a transformamos em um, dois, três, etc. Vamos implementar o padrão.

Construindo o interpretador

Nosso exemplo irá transformar um número romano informado para o nosso formato, arábico. Crie uma nova aplicação *Win32*. Deixe o formulário principal como na **Figura 2**.

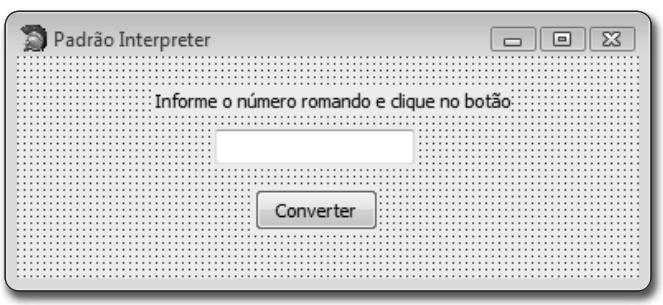


Figura 2. Formulário principal

Agora vamos criar uma classe que representa o número a ser convertido. Essa classe será chamada *TNumeral* e na **Listagem 4** vemos sua definição. A insira antes da declaração da classe *TForm1*.

Listagem 4. Classe TNume ral

```
01 TNumeral = class
02   private
03     FEntrada: string;
04     FSaida: integer;
05     procedure SetEntrada(const Value: string);
06     procedure SetSaida(const Value: integer);
07   public
08     constructor Create(NumeroRomano: string); reintroduce;
09     property Entrada: string read FEntrada write SetEntrada;
10     property Saida: integer read FSaida write SetSaida;
11 end;
```

Observe que na classe *TNumeral* temos duas propriedades, *Entrada* e *Saida*. *Entrada* receberá o algarismo romano que o usuário deseja converter e *Saida* irá armazenar o algarismo convertido. Passamos o valor a converter através do método *Create* da classe. Agora é preciso criar as classes que vão traduzir, converter, *interpretar* o algarismo romano. Vamos criar uma classe base chamada *TExpressao*, conforme **Listagem 5.**

Listagem 5. Classe TExpressao

```
01 TExpressao = class
02   public
03     function Um: string; virtual; abstract;
04     function Quatro: string; virtual; abstract;
05     function Cinco: string; virtual; abstract;
06     function Nove: string; virtual; abstract;
07     function Multiplicador: integer; virtual; abstract;
08
09     procedure Interpretar(Numeral: TNumeral);
10 end;
```

A interpretação em si se dá em identificar em que grupo numérico, milhar, centena, dezena ou unidade, o algarismo romano se enquadra e então através de um multiplicador gerar o número do grupo. Cada grupo numérico também será representado por uma classe. Veja que a classe *TExpressao* possui cinco funções marcadas como *virtual abstract*. Isso significa que sua implementação será feita em classes descendentes. Temos também o método *Interpretar*, ele é responsável por traduzir o algarismo romano. Na **Listagem 6** vemos seu código.

Listagem 6. Método Interpretar

```
01 procedure TExpressao.Interpretar(Numeral: TNumeral);
02 begin
03   if Trim(Numeral.Entrada) = '' then
04      raise Exception.Create('Número romano inválido');
05
06   if StartsStr(Nove,Numeral.Entrada) then
07   begin
08     Numeral.Saida := Numeral.Saida + (9 * Multiplicador);
09     Numeral.Entrada := Copy(Numeral.Entrada,3, Length(Numeral.
        Entrada));
10   end
11   else
12   if StartsStr(Quatro, Numeral.Entrada) then
13   begin
14     Numeral.Saida := Numeral.Saida + (4 * Multiplicador);
15     Numeral.Entrada := Copy(Numeral.Entrada,3, Length(Numeral.
        Entrada));
16   end
17   else
18   if StartsStr(Cinco, Numeral.Entrada) then
19   begin
20     Numeral.Saida := Numeral.Saida + (5 * Multiplicador);
21     Numeral.Entrada := Copy(Numeral.Entrada,2, Length(Numeral.
        Entrada));
22   end;
23
24   while StartsStr(Um, Numeral.Entrada) do
25   begin
26     Numeral.Saida := Numeral.Saida + (1 * Multiplicador);
27     Numeral.Entrada := Copy(Numeral.Entrada,2,
        Length(Numeral.Entrada));
28   end;
29 end;
```

No bloco de código contido entre as linhas 6 e 22 estamos verificando se o algarismo romano inicia com um determinado caractere que é retornado pelos métodos *Nove, Quatro e Cinco.* Esses métodos de acordo com o grupo numérico, *milhar, centena, dezena,* retornam o símbolo romano correspondente. Por exemplo se no momento o grupo numérico é *centena*, a função *Nove* retorna o algarismo romano correspondente a *900* e assim por diante. A função *StartsStr*, contida na *unit StrUtils*, verifica

se uma determinada *string* inicia com o texto informado. Neste caso verificamos se o algarismo inicia com o caractere romano retornado pelas funções. Se sim, somamos à propriedade *Saida* o multiplicador correspondente. Se o grupo numérico é *centena* esse multiplicador vale *100*. Então excluídos da leitura o que já foi lido, através da função *Copy*. No comando *while* seguinte terminamos a tradução do algarismo romano. As funções *Nove, Quatro* e *Cinco* acabam representando a mudança na estrutura representativa dos algarismos romanos, Observe a sequência: I, II, III, *IV, V,* VI, VII, VIII e *IX*. O caractere *I* é utilizado até o número *três,* então *V* é adicionado para a representação do número *quatro,* no *cinco* o *V* tem sua sequência iniciada e vai até o *nove,* quando o *X* é inserido. Isso se dá em todos os grupos numéricos, da *unidade* ao *milhar.* Vamos agora criar as outras classes que representam os grupos numéricos. Na **Listagem 7** vemos a classe *TExpressaoMilhar.*

Listagem 7. Classe TExpressaoMilhar

```
01 TExpressaoMilhar = class(TExpressao)
02  public
03  function Um: string; override;
04  function Quatro: string; override;
05  function Cinco: string; override;
06  function Nove: string; override;
07  function Multiplicador: integer; override;
08 end;
```

Todas as classes que representam os grupos numéricos herdam de *TExpressao* e implementam suas funções, como pode ser visto na **Listagem 8**.

Listagem 8. Implementando o grupo numérico milhar

```
01 function TExpressaoMilhar.Multiplicador: integer;
02 begin
03   result := 1000;
04 end;
05
06 function TExpressaoMilhar.Nove: string;
07 begin
08   result := ' ';
09 end;
10
11 function TExpressaoMilhar.Quatro: string;
12 begin
13  result := ' ';
14 end;
15
16 function TExpressaoMilhar.Um: string;
17 begin
18   result := 'M';
19 end;
```

O que é importante ser destacado aqui é que a função *Multiplicador* retorna o valor *1000,* que corresponde ao seu grupo e a outra função *Um* retorna o valor correspondente em romano. Na **Listagem 9** vemos a definição da da classe *TExpressaoCentena* e sua implementação.

Como pode ser visto agora os métodos *Um, Quatro, Cinco e Nove* estão retornando os caracteres romanos apropriados, enquanto que mais uma vez o método *Multiplicador* retorna o grupo correspondente. O mesmo se dará para os dois grupos restantes, *dezena e unidade*, que estão representados pelas classes *TExpressaoDezena* e *TExpressaoUnidade* que por questões de espaço não serão listadas aqui.

Listagem 9. Implementando o grupo numérico centena

```
01 TExpressaoCentena = class(TExpressao)
02 public
03   function Um: string; override;
04   function Quatro: string; override;
05   function Cinco: string; override;
06   function Nove: string; override;
07   function Multiplicador: integer; override;
08 end;
09 function TExpressaoCentena.Cinco: string;
10 begin
11   result := 'D';
12 end;
13
14 function TExpressaoCentena.Multiplicador: integer;
15 begin
16   result := 100;
17 end;
18
19 function TExpressaoCentena.Nove: string;
20 begin
21   result := 'CM';
22 end;
23
24 function TExpressaoCentena.Quatro: string;
25 begin
26   result := 'CD';
27 end;
28
29 function TExpressaoCentena.Um: string;
30 begin
31   result := 'C';
32 end;
```

Com tudo isso pronto só falta realizar a conversão em si. Isso acontece no *OnClick* do botão, como pode ser visto na **Listagem 10**.

Listagem 10. Realizando a conversão

```
01 procedure TForm1.Button1Click(Sender: TObject);
02 var
03   NumeroRomano : TNumeral;
04   Lista: array[1..4] of TObject;
05   x: integer;
06 begin
07   NumeroRomano := TNumeral.Create(EditRomano.Text);
08   Lista[1] := TExpressaoMilhar.Create;
09   Lista[2] := TExpressaoCentena.Create;
10   Lista[3] := TExpressaoDezena.Create;
11   Lista[4] := TExpressaoUnidade.Create;
12
13   for x:= 1 to 4 do
14   begin
15     TExpressao(Lista[x]).Interpretar(NumeroRomano);
16   end;
17
18   ShowMessage('O número convertido é: ' +
       IntToStr(NumeroRomano.Saida));
19
20   FreeAndNil(NumeroRomano);
21 end;
```

Para realizar a conversão criamos um *array* de *TObject* que irá armazenar os quatro grupos numéricos. Depois na linha 7 criamos instanciamos a classe *TNumeral*, passando no seu *Create* o algarismo romano digitado pelo usuário. Então em cada posição do *array* instanciamos um grupo numérico, partindo do maior para o menor. No comando *for* da linha 13 realizamos um *cast*. Isso é necessário porque os objetos de tipo *TExpressaoMilhar, TExpressaoCentena, TExpressaoDezena* e *TExpressaoUnidade* foram armazenados como *TObject*. Isso foi possível porque em *Delphi* toda classe tem como ancestral comum o *TObject*. A operação de *cast* tenta converter um determinado tipo para outro. Se a conversão não for possível uma exceção é levantada. Em nosso caso estamos convertendo de

objetos do tipo *TObject* para o tipo *TExpressao*. Isso é possível porque as classes de expressão que foram adicionadas no *array* herdam de *TExpressao*. Com o *cast* realizado, então chamamos o método *Interpretar* que faz a conversão e exibimos o resultado na tela utilizando um *ShowMessage*. Execute a aplicação debugando o commando *for* para que você possa entender com clareza os recursos de polimorfismo e herança utizados para a construção do padrão *interpreter*. Como já foi dito, implementar um interpretador é uma tarefa complexa que exige uma solução complexa.

Padrão Memento

O intento do padrão *Memento* é oferecer meios de armazenamento e restauração do estado de um objeto. Esse armazenamento pode ser realizado em memória, arquivo, ou até mesmo banco de dados. O *Delphi* aplica esse padrão nos objetos *TDataSet*. Isso pode ser visto quando inserimos um registro, preenchemos seus campos, mas, ao invés de salvar os dados realizamos um *Cancel*, que restaura para nós o *dataset* em seu estado anterior. É possível aplicar esse padrão em outras circunstâncias também, por exemplo, para restaurar o último estado de um formulário. Vamos ver como implementar isso.

Nota

Existem componentes de terceiros que já oferecem esse recurso, como o pacote JVCL.

Inicie uma nova aplicação *Win32*. Neste exemplo vamos salvar e restaurar de um arquivo *.INI* o tamanho de um formulário. Nessa nova aplicação adicione uma nova *unit*, chamando-a de *Memento.pas*. Na seção *uses* adicione uma referência a *IniFiles*, *Forms* e *SysUtils*. Agora vamos criar a classe *TEstadoForm* conforme **Listagem 11**.

Listagem 11. Classe TEstadoForm

```
01  TEstadoForm = class
02  private
03    FFormCorrente: TForm;
04  public
05    constructor Create(Form: TForm); reintroduce;
06    procedure RecuperarEstado;
07    procedure SalvarEstado;
08  end;
```

Nessa classe nós temos o construtor que recebe como parâmetro o formulário que terá seu tamanho salvo. As operações para restaurar e salvar estão representadas nos métodos *RecuperarEstado* e *SalvarEstado*, que podem ser vistos na **Listagem 12**.

Como pode ser visto estamos utilizando um arquivo *.INI* para registrar as propriedades *Width* e *Height* de um formulário. Esse arquivo é criado na pasta onde o executável é localizado e possui como nome, o nome do formulário que será armazenado. No formulário principal do projeto adicione dois botões, um com o *Caption Salvar* e outro *Restaurar*. Adicione uma referência à *unit* que criamos, *Memento.pas*. Na seção *private* desse formulário crie uma variável do tipo *TEstadoForm* e no evento *OnCreate* instancie essa variável como no código a seguir.

```
FMemento := TEstadoForm.Create(Self);
```

Ao passarmos *Self* no construtor do objeto estamos passando uma referência ao próprio formulário. Posteriormente, no evento *OnClick* do botão *Salvar* insira o seguinte código:

```
FMemento.SalvarEstado;
```

E no *OnClick* do botão *Restaurar* o código a seguir.

```
FMemento.RecuperarEstado;
```

Execute a aplicação e primeiramente clique no botão *Salvar*. Depois, redimensione o formulário e clique no botão *Restaurar*. O *estado* do formulário será recuperado. Esse é o padrão *Memento*.

Outros padrões

Até o presente momento vimos os chamados *padrões de arquitetura de software*. Porém existem outros padrões denominados *padrões de arquitetura para aplicações empresariais*. Esses padrões são descritos por Martin Fowler em seu livro *Patterns of Enteprise Application Architecture*. Martin Fowler percebeu que, independentemente da liguagem utilizada, os *desenvolvedores* de sistemas para empresas, enfrentam os mesmos problemas e geralmente se utilizam das mesmas soluções. Essas soluções foram catalogadas e tornaram-se padrões. Vamos ver alguns deles, suas definições e responsabilidades. Não cobrirei aqui como implementá-los, visto que para isso seria necessário outra série de artigos.

Padrão ActiveRecord

Este pertence a uma categoria chamada *Padrões de Arquitetura para Fontes de Dados*. Nessa categoria são relacionadas soluções para o problema de *como* manipular uma fonte de dados, seja ela *SQL*, arquivos *XML* ou outros, da forma mais simples possível . O *Active Record* é um objeto que encapsula

Listagem 12. Salvando e recuperando

```
01 procedure TEstadoForm.RecuperarEstado;
02 var
03  Arquivo: TINIFile;
04 begin
05   Arquivo := TINIFile.Create(ExtractFilePath(Application.
      ExeName)+ FFormCorrente.Name + '.ini');
06   FFormCorrente.Width := Arquivo.ReadInteger('FORMULARIO',
      'Width', 800);
07   FFormCorrente.Height := Arquivo.ReadInteger('FORMULARIO',
      'Height', 600);
08  Arquivo.Free;
09 end;
10
11 procedure TEstadoForm.SalvarEstado;
12 var
13  Arquivo: TINIFile;
14 begin
15   Arquivo := TINIFile.Create(ExtractFilePath(Application.
      ExeName)+ FFormCorrente.Name + '.ini');
16   Arquivo.WriteFloat('FORMULARIO','Width', FFormCorrente.
      Width);
17   Arquivo.WriteFloat('FORMULARIO','Height', FFormCorrente.
      Height);
18   Arquivo.UpdateFile;
19   Arquivo.Free;
20 end;
```

um registro de uma tabela de um banco de dados, ou outra fonte, mas que também encapsula seu acesso e adiciona lógica do domínio sobre os dados. De certa forma todos nós usamos esse padrão e não percebemos. Ele está nativamente no *Delphi* através dos objetos *TClientDataSet*. Utilizamos esses objetos para representar *entidades* do nosso negócio, como *Produto* ou *Fornecedor*. Podemos inserir sobre esses dados toda lógica de validação além é claro, ter a possibilidade de incluir, alterar ou remover a *entidade*.

Padrão Lazy Load

Este é utilizando quando programamos de forma orientada a objetos, quando utilizamos uma ferramenta de persistência para salvar e recuperar os objetos. Não trabalhamos diretamente com *TClientDataSet*. Temos objetos reais que representam o domínio. *Lazy load* significa carregamento tardio. Trazendo isso para o contexto objeto relacional, corresponde à situação onde temos um objeto *NotaFiscal* que possui uma associação com objetos do tipo *Produto*. Ou seja, possui um coleção de objetos *Produto*. Imagine que sempre que você precisar acessar alguma informação de um objeto *NotaFiscal* e instanciá-lo, sua lista de produtos também será carregada. Isso pode gerar um tráfego de informações desnecessário visto que você precisa apenas, por exemplo, da data da nota fiscal. Então se criou o padrão *lazy load* que diz que essas coleções só são carregadas quando se faz o acesso a elas.

Padrão Repository

Este é um padrão sofisticado que é composto por outros padrões, mas essencialmente o *Repository* encapsula um conjunto de objetos persistidos em algum meio, e oferece recursos para localização desses objetos. Seu pode deixar seu código mais claro quando várias pesquisas precisam ser utilizadas, isso porque reduz a quantidade de código. Apenas para facilitar o entendimento, imagine que exista em algum sistema seu uma tabela *Produtos*. Constantemente é preciso localizar os dados do produto e sempre eu seu código você insere sentenças SQL para isso. Para aplicar parcialmente esse padrão, de forma bem simplista, criamos uma classe *RepositorioProdutos* e nela criamos métodos para localização dos produtos.

Conclusão

No nosso dia a dia trabalhamos de diferentes formas. Às vezes nos indagamos se somos os únicos a aplicar uma determinada solução para um problema, ou se estamos fazendo *certo, errado.* Ao ler sobre padrões de projetos, não só sobre os de arquitetura, mas também sobre os empresariais, podemos ter uma idéia se estamos no caminho certo ou errado. Foram apresentados ao longo desses três artigos alguns padrões e sua aplicação para o dia a dia, cabe agora a você investigar os demais e aplicar as boas práticas que garantem um software de melhor manutenção. Recomendo a leitura do livro *Patterns of Enterprise Application Architecture*, que já possui tradução para o português e do livro *Design Patterns Explained – A New Perspective on Object Oriented Design*. Boa leitura! ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos sobre técnicas que poderão aumentar a qualidade do desenvolvimento de software

tiOPF na Prática

Faça um controle de cheques usando orientação a objetos

Resumo do DevMan

O tiOPF é um dos mais antigos e estáveis frameworks ORM, Object/Relational Mapper, para Delphi. É utilizado por grandes empresas no exterior e, neste artigo vamos aplicá-lo no desenvolvimento de um sistema simples, mas que representa o dia a dia de qualquer desenvolvedor. Vamos abordar inclusão, consulta, exclusão e até mesmo geração de relatórios.

Nesse artigo veremos
- Uma introdução ao framework tiOPF;
- Como utilizá-lo em um sistema comum.

Qual a finalidade
Mostrar o uso adequado de ferramentas que auxiliam no desenvolvimento orientado a objetos.

Quais situações utilizam esse recurso?
No desenvolvimento de sistemas em geral.

Paulo Quicoli
pauloquicoli@devmedia.com.br
Editor técnico da revista ClubeDelphi, formado em processamento de dados pela FATEC-TQ. Atua como analista e desenvolvedor da ControlM Informática (www.controlm.com.br). Utiliza Delphi desde sua primeira versão, entusiasta do desenvolvimento orientado a objetos tem publicado vários artigos sobre o assunto. Blog: www.pauloquicoli.spaces.live.com .

O *tiOPF* é um *framework* que realiza o mapeamento objeto relacional para nós, assumindo a responsabilidade de persistir objetos de negócio em tabelas de um banco de dados. Quando desenvolvemos um sistema aplicando a orientação a objetos nos deparamos com um problema chamado *impendância objeto/relacional*. Resumidamente esse problema representa a diferença existente entre esses dois mundos.

Para exemplificar a situação, imagine que temos duas classes, *TPessoa* e *TEndereco*. Elas são relacionadas uma com a outra, ou seja, um objeto *TPessoa* possui um objeto *TEndereco*. Agora pense em como essas informações poderiam ser salvas em um banco de dados relacional.

Poderíamos ter uma tabela *PESSOA* e outra *ENDERECO* ligadas por uma chave estrangeira e ainda poderíamos ter apenas uma tabela *PESSOA* que conteria todos os campos necessários. Uma ferramenta *ORM* cuida de tudo isso para nós, permitindo que mantenhamos nosso foco no que fazemos de melhor, desenvolver realmente o sistema.

Instalando o tiOPF

A primeira coisa que precisamos fazer é realizar seu *download*, que é gratuito e ainda com código fonte disponível. Acesse *http://tiopf.sourceforge.net/* e realize o *download*. Na sequência execute seu instalador. Durante a instalação existe um ponto importante, visto na **Figura 1**.

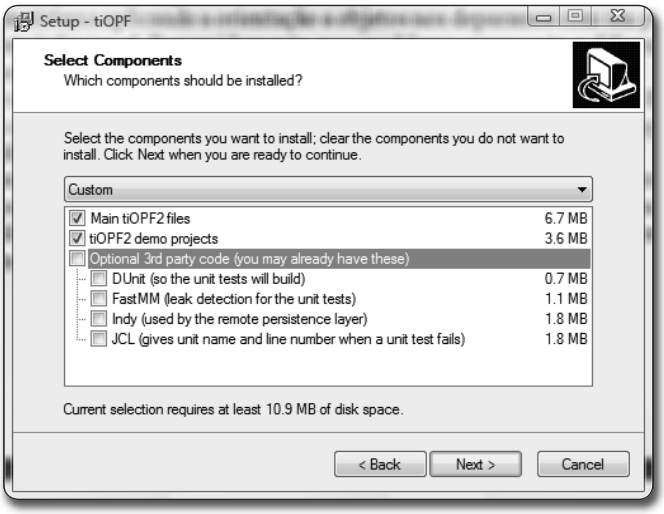


Figura 1. Instalação di tiOPF

Ao se deparar com essa tela devemos escolher quais componentes serão instalados. Aqui eu já possuo todos o opcionais, *DUnit*, *Fast MM*, *Indy*, por isso os deixei desabilitados. Deixe marcado apenas os que você ainda não instalou.

O instalar apenas descompacta o código fonte na pasta selecionada, sendo necessário agora instalar os pacotes no *Delphi*. Abra a pasta de instalação do *tiOPF* e acesse a pasta *Compilers*. Nela teremos pacotes para o *Delphi 7, Delphi 2006, Delphi 2007* e *FreePascal*. Abra a pasta de acordo com a versão de *Delphi* que você utiliza e abra o *Project group* correspondente. No meu caso vou utilizar o *Delphi 2007*, portando abro o arquivo *tiOPF_D2007.groupproj*.

Agora é necessário adicionar no *Library Path* o caminho para as pastas *\tiOPF2\Compilers\Delphi2007* , *\tiOPF2\Core*, *\tiOPF2\Gui e \tiOPF2\Options*. Então podemos efetuar um *build* nos projetos *tiOPFCore2007.bpl*, *tiOPFOptions2007.bpl*, *tiOPFGUI2007.bpl e tiOPFGUIDsgn2007.bpl*. Neste último realizamos um *install* também porque ele irá instalar os controles visuais do *tiOPF*, como na **Figura 2**.

Planejando o sistema

Nosso sistema será um controle de cheques emitidos, bem simples. Poderemos nele cadastrar várias contas bancárias e então cadastrar os cheques que emitimos, informando a data para depósito, valor e porque foi emitido. Esse cadastro será utilizado para responder questões como quantos cheques serão depositados em determinado mês, quantos foram emitidos. Também vamos emitir relatórios dessas pesquisas.

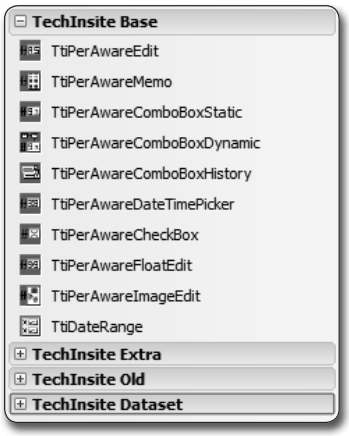


Figura 2. Controles visuais

Digrama de classes

Como vamos aplicar orientação a objetos e vamos começar o projeto *do zero*, modelamos inicialmente o digrama de classes que representa os objetos de negócio, como pode ser visto na **Figura 3**.

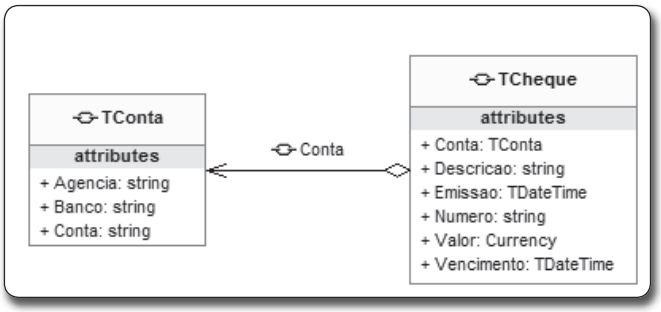


Figura 3. Classes de negócio

Temos então duas classes, *TConta* e *TCheque*. *TCheque* possui uma referência à *TConta*.

Implementando o diagrama usando tiOPF

Vamos codificar nossas classes de negócio para serem persistidas pelo *framework tiOPF*. Crie um novo projeto do tipo *Win32* e nele adicione uma nova *unit*. Vamos chamá-la de *BOM.pas*, sigla para *Business Object Model* ou, *Modelo de Objetos de Negócio*.

O *tiOPF* já oferece para nós uma estrutura básica pronta para inserção, exclusão e consulta de objetos, basta seguirmos essa estrutura. Uma das regras é que toda classe que será salva em um meio físico precisa herdar de *tiObject*. Outra regra é que as propriedades que serão salvas devem estar declaradas na seção *published*. Isto porque internamente é feito o uso da *RTTI, RunTime Type Information*, do *Delphi* para que o *framework* possa identificar o tipo de uma propriedade e saber como inserir seu valor no banco de dados. Essas operações são chamadas de *CRUD, Commit – Retrieve – Update – Delete*, e são realizadas através de *coleções de objetos*, o que implica que cada classe deve ter sua lista. Vamos ver isso a seguir.

Na *unit BOM.pas* vamos adicionar nossas primeiras classes, *TConta* e *TContas*. Comecemos pela coleção. Veja **Listagem 1**.

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora mesmo o portal do assinante ClubeDelphi e veja uma vídeo aula de Guinther Pauli como utilizar corretamente o rtti (runtime type information) no delphi.

*http://www.devmedia.com.br/articles/viewcomp.asp?comp=987&hl=*RTTI**

Listagem 1. Classes TContas

```
01 type
02   TContas = class;
03   TConta = class;
04
05   TContas = class(TtiObjectList)
06   private
07   protected
08     function GetItems(i: integer): TConta; reintroduce;
09     procedure SetItems(i: integer; const Value: TConta); reintroduce;
10   public
11     property Items[i:integer]: TConta read GetItems write SetItems;
12     procedure Add(AObject: TConta); reintroduce;
13     function  Find(pOIDToFindAsString: string): TConta;  reintroduce;
14     procedure Clear; override;
15     procedure Read; override;
16     procedure Save; override;
17   end;
```

Nas linhas 2 e 3 declaramos antecipadamente as duas classes que usuremos. Veja na linha 5 que criamos a classe *TContas* herdando de *TtiObjectList*, que é base para lista de objetos oferecendo métodos para salvar e carregar as informações. Sua implementação também é bem simples, como mostra a **Listagem 2**.

Como essa classe é uma coleção de objetos, nada mais lógico do que ela oferecer meios para se adicionar nela esses objetos. É exatamente isso que o método *Add* faz. O método *Clear* por sua vez marca o estado a coleção em *posEmpty*, que indica ao *framework* que a mesma deve ser esvaziada. Se olharmos agora o *Find* veremos uma ação de recuperação de objetos. Nesse caso vamos passar o *OID, object identification*, de um objeto qualquer do tipo *TConta* e o mesmo será localizado e retornado. Quando herdamos nossas classes de *TtiObject* trazemos já essa propriedade, *OID*, que é a identificação única do objeto em questão, ou seja, é como uma chave primária. Nesta demonstração vamos utilizar chaves primárias do tipo *GUID*, que nada mais são do que uma *string* de 36 posições que é gerada por mecanismos do sistema operacional *Windows* que garantem sua unicidade. Para salvar os objetos da lista temos o método *Save*, que apenas executa o código herdado e, para carregar todas as informações na lista temos o *Read*. Chegou a hora de criarmos *TConta*. Veja isso na **Listagem 3.**

Listagem 2. Implementando TContas

```
01 procedure TContas.Add(AObject: TConta);
02 begin
03   inherited Add(AObject);
04 end;
05
06 procedure TContas.Clear;
07 begin
08   inherited;
09   ObjectState:= posEmpty;
10 end;
11
12 function TContas.Find(pOIDToFindAsString: string): TConta;
13 begin
14   if pOIDToFindAsString = '' then
15     result:= nil
16   else
17     result:= TConta(inherited Find(pOIDToFindAsString));
18 end;
19
20 function TContas.GetItems(i: integer): TConta;
21 begin
22  result:= TConta(inherited GetItems(i));
23 end;
24
25 procedure TContas.Read;
26 begin
27   inherited;
28 end;
29
30 procedure TContas.Save;
31 begin
32   inherited;
33 end;
34
35 procedure TContas.SetItems(i: integer; const Value: TConta);
36 begin
37   inherited SetItems(i, Value);
38 end;
```

Listagem 3. Classe TConta

```
01  TConta = class(TtiObject)
02  private
03    FAgencia: string;
04    FConta: string;
05    FBanco: string;
06    procedure SetAgencia(const Value: string);
07    procedure SetConta(const Value: string);
08    procedure SetBanco(const Value: string);
09    function GetContaAsGui: string;
10  protected
11    function GetOwner: TContas; reintroduce;
12    procedure SetOwner(const Value: TContas); reintroduce;
13  public
14    constructor Create; override;
15    destructor Destroy; override;
16    property Owner: TContas read GetOwner write SetOwner;
17  published
18    property Agencia: string read FAgencia write SetAgencia;
19    property Conta: string read FConta write SetConta;
20    property Banco: string read FBanco write SetBanco;
21    property ContaAsGui: string read GetContaAsGui;
22  end;
```

Observe que as propriedades estão na seção *published*, contudo temos uma na seção *public*. A propriedade *Owner* é utilizada para indicar em qual lista de objetos ele está inserido. Temos também a propriedade *ContaAsGui* que é somente leitura. Ela foi criada para facilitar a exibição dos dados em um formulário. Seu valor de retorno é a soma das propriedades *Banco, Agencia e Conta*, como pode ser visto no código a seguir.

```
function TConta.GetContaAsGui: string;
begin
  result := FBanco + ' ' + FAgencia + '-'+ FConta;
end;
```

Vamos criar agora o banco de dados e tabela que irá conter os dados das contas. Crie um novo banco de dados *Firebird*, salvando-o como *Cheques.fdb*. Usando seu gerenciador preferido crie a tabela a seguir, como mostra **Listagem 4**.

Agora que temos a classe pronta e seu banco de dados pronto, é necessário informar isso ao *tiOPF*. Essa operação é chamada mapeamento objeto relacional. Vamos criar um procedimento global chamado *procedure RegisterAutoMappings* e na **Listagem 5** temos seu código.

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora mesmo o portal do assinante ClubeDelphi e veja uma vídeo aula de Luciano Pimenta que mostra como utilizar o IBExpert para administração avançada de banco de dados Firebird/Interbase.

http://www.devmedia.com.br/articles/viewcomp.asp?comp=3082

Listagem 4. Script SQL

```
CREATE TABLE CONTA (
    AGENCIA    VARCHAR(15),
    NRO_CONTA  VARCHAR(15),
    BANCO      VARCHAR(50),
    ID_CONTA   VARCHAR(36) NOT NULL
);
ALTER TABLE CONTA ADD CONSTRAINT PK_CONTA PRIMARY KEY (ID_
CONTA);
```

Listagem 5. Realizando o mapeamento

```
01 procedure RegisterAutoMappings;
02 begin
03   with GTIOPFManager.ClassDBMappingMgr do
04   begin
05     RegisterMapping(TConta,'CONTA','OID', 'ID_CONTA', [pktDB]
);
06     RegisterMapping(TConta,'CONTA','Agencia','AGENCIA');
07     RegisterMapping(TConta,'CONTA','Conta','NRO_CONTA');
08     RegisterMapping(TConta,'CONTA','Banco','BANCO');
09     RegisterCollection(TContas, TConta);
10 end;
11 end;
```

Nota

Adicione na seção uses as seguintes referências: tiOPFManager, tiAutomap, tiObject, tiOID, tiQuery e tiQueryIBX.

O *tiOPF* possui uma espécie de gerenciador de objetos representado pela instância *GTIOPFManager* e nela podemos indicar em qual tabela e em quais campos o a referida classe é salva, como pode ser visto no método *RegisterMapping* que recebe como parâmetro o tipo da classe a ser salva, o nome da tabela que receberá a classe, o nome da propriedade que será salva e o nome do respectivo campo. Veja que nesse mapeamento não incluímos a propriedade *ContaAsGui*. Ou seja, se você possui uma propriedade calculada, como no nosso caso, e a mesma não precisa ser salva, apenas não a inclua no mapeamento.

Criando a interface do usuário

Como já temos nossa primeira classe pronta para ser salva, vamos criar a interface do programa então. No formulário principal adicione *TMainMenu*, criando os itens de menu *Cadastros>Contas* e *Consultas>Cheques*. Depois adicione um *DataModule* e adicione referências às *units BOM* e *tiOPFManager*. No seu evento *OnCreate* vamos *startar* o *framework* com o código a seguir.

```
GTIOPFManager.DefaultPersistenceLayerName:='IBX';
GTIOPFManager.ConnectDatabase(
        'localhost:C:\Paulo\ClubeDelphi\Artigos\tiOPF\Demo\Banco\
          cheques.fdb',
        'SYSDBA','masterkey');
RegisterAutoMappings;
```

Estamos definindo qual camada de persistência, componente de acesso a banco, será utilizada. Optei pela *IBX*, contudo o *tiOPF* oferece suporte aos seguintes componentes:

- Interbase/Firebird via IBX ;
- Firebird via FBLib;
- Firebird via ZeosLib;
- Oracle via DOA;
- MS Access e MS SQL-Server via ADO;
- Paradox via BDE;
- XML via MSDOM ou DOM do FreePascal;
- Arquivos CSV.

Depois informamos os parâmetros para conexão do banco, através do método *ConnectDatabase* que já realiza a conexão para então chamar o método *RegisterAutoMappings*. Como já foi mencionado o *tiOPF* nos oferece uma gama de componentes que podemos utilizar para facilitar esse trabalho. Nesse momento vamos criar o cadastro de contas.

Cadastrando uma conta

Adicione um novo formulário ao projeto, salvando-o como *cadContaU.pas*. Nele vamos adicionar uma *TToolbar*, um *TImageList* para ser ligado a esse *TToolbar* e nela vamos adicionar um botão *Salvar*. Depois vamos inserir um controle do tipo TtiVTListView que está na guia *TechInsite Extra*. Altere as propriedades *EditInLineOnly* para *false*, *VisibleButtons.tiLVTBtnVisEdit* para *true*, *VisibleButtons.tiLVTBtnVisNew* para *True*, *VisibleButtons.tiLVTBtnVisDelete* para *true*, *Name* para *ListView* e *Header.Style* para *hsXPStyle*. O formulário deve ficar como na **Figura 4**.

Esse *ListView* é especializado para ser utilizado com objetos do *tiOPF*, por isso ele possui alguns eventos especiais que vamos fazer uso mais tarde. Adicione o no formulário, na seção *private* um campo *FContas* do tipo *TContas*. No evento *OnCreate* do formulário adicionamos o código da **Listagem 6.**

Ao criar o formulário instanciamos a lista de objetos e então criamos as colunas no *ListView*. O método *AddColumn* recebe como parâmetro o nome da propriedade que será exibida no controle, qual o tipo dessa propriedade, o *caption* do cabeçalho da coluna e o seu tamanho.

Depois carregamos todas as contas já cadastras utilizando o método *Read*. Então para que esses dados sejam exibidos, os ligamos ao *ListView* através da propriedade *Data*. Acredito que ficou claro que até agora que não escrevemos nenhum comando SQL dentro da aplicação para recuperar os dados da tabela *CONTA*. Vamos fazer uso dos eventos especiais

Figura 4. Cadastro de contas

Listagem 6. Carregando as contas

```
01  FContas := TContas.Create;
02  ListView.AddColumn('Banco', vttkString, 'Banco', 150);
03  ListView.AddColumn('Conta', vttkString, 'Conta', 150);
04  ListView.AddColumn('Agencia', vttkString, 'Agência', 150);
05
06  ListView.Data:= nil;
07  FContas.Clear;
08  Fcontas.Read;
09  ListView.Data:= FContas;
```

Listagem 7. Ligando os dados do objeto

```
01 public
02     { Public declarations }
03     procedure SetData(const Value: TtiObject); override ;
04 end;
05
06 {...}
07
08 procedure TEditContaF.SetData(const Value: TtiObject);
09 begin
10  inherited;
11   //Ligando os dados
12   editBanco.LinkToData(DataBuffer,'Banco');
13   editAgencia.LinkToData(DataBuffer,'Agencia');
14   editConta.LinkToData(DataBuffer,'Conta');
15 end;
```

Listagem 8. Incluindo uma nova conta

```
01 procedure TcadContaF.ListViewItemInsert(pVT: TtiCustomVirtualTree;
02   AData: TtiObject; AItem: PVirtualNode);
03 var
04   obj : TConta ;
05 begin
06   obj:= TConta.CreateNew;
07   obj.Owner := FContas;
08   if EditContaF.Execute(obj) then
09   begin
10     FContas.Add(obj);
11     ListView.Refresh(obj);
12   end
13   else
14    obj.Free ;
15 end;
```

desse controle. Temos o evento *OnItemDelete*. Ele é disparado quando o usuário tenta excluir um objeto da lista. Nesse evento inserimos o seguinte código:

```
if tiPerObjAbsConfirmAndDelete(AData) then
begin
  FContas.Save;
  pVT.Refresh;
end;
```

A função *tiPerObjAbsConfirmAndDelete* também já vem pronto no pacote *tiOPF*. É uma caixa de diálogo que solicita a confirmação para a exclusão. Se houve a confirmação, o objeto que foi excluído é marcado para exclusão e ao chamarmos *FContas.Save* ele é realmente excluído do banco de dados. Note que anteriormente eu disse que o objeto é marcado para exclusão. Para que objetos nessa situação não possam ser visualizados no *ListView*, em seu evento *OnFilterData* inserimos o seguinte código:

```
pInclude := not aData.Deleted;
```

Isso faz com que os objetos deletados não se tornem visíveis. Para inserir ou editar um objeto *TConta* vamos utilizar um segundo formulário. Dessa vez esse formulário irá herdar de um básico vindo com o *framework*. Adicione ao projeto a *unit FtiPerEditDialog*, localizada na pasta *GUI* da instalação do componente. Crie outro formulário que herde deste. Salve-o como *EditContaU.pas*. Nele vamos adicionar três controles *TtiPerAwareEdit*. Estes também são especializados para uso de objetos *tiObject*. Altere seus nomes para *editBanco, editAgencia* e *editConta*. Depois sobrescrevemos o método *SetData* como a seguir, **Listagem 7**.

Como pode ser visto, os controles de edição oferecem meios para que possamos ligar as propriedades. Esse método *SetData* é utilizado pelo formulário pai.

Voltando agora ao formulário *CadContaF*, vamos implementar o evento *OnItemInsert* como mostra **Listagem 8.**

Nas linhas 6 e 7 criamos um novo objeto *Conta* e já o relacionamos com a lista *FContas*. Depois, chamamos nosso formulário de edição de contas. Veja que é executado através do método *Execute* que aceita como parâmetro um objeto *TtiObject*. Se lá na tela de edição o usuário escolheu o *OK*, é retornado *true* e então adicionamos o objeto na lista *FContas* e então pedimos pro *ListView* atualizar a exibição desse objeto. A edição também é semelhante, como código a seguir lançado no evento *OnEditItem*.

```
procedure TcadContaF.ListViewItemEdit(pVT: TtiCustomVirtualTree;
  AData: TtiObject; AItem: PVirtualNode);
begin
  if EditContaF.Execute(AData) then
    pVT.Refresh(AData);
end;
```

Até o presente momento não enviamos nada para o banco de dados, ou seja, não *comitamos* as alterações. Como já foi dito, isso é tarefa das listas de objetos, portando no botão *Salvar* codificamos apenas:

```
FContas.Save;
```

No formulário principal, no item de menu *Cadastros>Contas* apenas chamamos o formulário *TCadContaF*. Execute a aplicação e veja que estamos adicionando registros, editando, excluindo, sem nos preocupar sentença alguma de *SQL*.

Conclusão

Quero destacar aqui a total ausência de comandos SQL na aplicação. Isso nos permitiu manter o foco na aplicação e não em rotinas de inserção, exclusão, etc. No próximo artigo daremos continuidade ao projeto, realizando o cadastro dos cheques, consulta e impressão de listagem. Não perca! ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Você não está mais sozinho.

A partir de agora você pode contar com a ajuda da

Chegou a Consultoria On-line DevMedia

Consultoria Técnica + Professor Virtual + Certificação

Mais Informações:
www.devmedia.com.br/consultoria_online 21 3382-5025

DevMedia em seus projetos e estudos.

A DevMedia possui um numeroso time de autores, editores e professores que juntos produzem o material que você está acostumado a encontrar em nosso site e revistas. E são exatamente esses mesmos profissionais que estarão a sua disposição para tirar suas dúvidas e ajudá-lo em seus projetos e estudos. Através de uma plataforma 100% web a Consultoria DevMedia garante sigilo absoluto, eficiência e rapidez em todas as respostas. Finalmente você terá ao seu alcance uma consultoria de qualidade por um preço muito acessível. Consulte nossos planos.

Mais um serviço DevMedia group

Nesta seção você encontra artigos para iniciantes na linguagem Delphi

Introdução ao Ribbon Control do Delphi 2009

Conheça os novos componentes da VCL

Nesse artigo veremos

• Introdução ao Ribbon Controls, a nova paleta de componentes do Delphi 2009;

• Sua utilização em conjunto com o componente Action Manager.

Qual a finalidade

Demonstrar o uso do componente Ribbon Control em uma aplicação real.

Quais situações utilizam esse recurso?

Para qualquer aplicação que se deseja aperfeiçoar o visual e a interação com o usuário, fazendo uso do que há de mais novo em GUI, Graphical User Interface.

Fabrício Hissao Kawata

fabricio.kawata@bol.com.br

Natural de Araçatuba-SP, formado em Processamento de Dados pela FATEC-TQ. Analista Programador (Delphi e Java) há 3 anos, sendo 2 destes em uma empresa do ramo de telecomunicações.

Resumo do DevMan

Com o lançamento do pacote Microsoft Office 2007 notou-se que seus aplicativos, por exemplo Word e Excel, traziam como maior novidade visual a ausência dos menus e barras de ferramentas. Eles foram substituídos por comandos organizados em guias, cada qual com funcionalidades relevantes para a ação a ser realizada, dentro de uma faixa de opções, ou Ribbon em inglês. Tal mudança teve como principal objetivo agilizar o trabalho do usuário, dando mais opções para sua interação com o aplicativo.

A utilização desse controle não é difícil e exige apenas algumas horas de estudo para realizarmos as mais diversas tarefas em nossa aplicação. Veremos então como utilizar o componente na prática.

A mais nova versão do *RAD Studio*, o *Delphi 2009*, trouxe inúmeras novidades em sua *IDE*. Na parte *VCL* foi adicionada uma nova paleta de componentes denominada *Ribbon Controls*, aqueles mesmos usados nos aplicativos do *Microsoft Office 2007*. Tais elementos têm como foco melhorar a parte de experiência com o usuário, abordando um novo conceito de *GUI* para aplicações.

Nota do DevMan

GUI - Graphics User Interface, ou interface gráfica, em português, é o mecanismo de interação entre o computador e seu usuário baseado em símbolos visuais, como botões, ícones, menus e janelas. Através de um dispositivo de entrada (ex: mouse e teclado) o usuário é capaz de acionar esses símbolos e manipulá-los de forma a obter algum resultado prático.

A psicologia cognitiva, que estuda o processo mental que está por detrás do comportamento humano, afirma que o nosso cérebro reage com maior eficácia ao se visualizar elementos gráficos se comparado à assimilação de palavras.

Neste artigo, para fins didáticos, abordaremos esses novos componentes através de um exemplo prático. Será desenvolvido um editor de textos simples com foco na utilização dos principais elementos da paleta *Ribbon Controls*. As funcionalidades do aplicativo em si, ficarão em segundo plano, já que este não é o objetivo da matéria.

Essa abordagem inicial permitirá que você se familiarize com tais componentes mostrando seus recursos básicos e deixando sua mente aberta para o seu uso em outros tipos de aplicações. Tudo será feito passo-a-passo, partindo da instalação da nova *IDE* da *Embarcadero*.

Preparação do ambiente de desenvolvimento

Certamente nosso primeiro passo será acessar o site da *Embarcadero* para fazermos o *download* do *RAD Studio Delphi 2009*. A versão está disponível para teste de 30 dias, Trial. Então, utilizando seu navegador, entre em *www.embarcadero.com* e vá para a seção *Products<RAD Studio* e então clique em *Download Trial*. Ao entrar nesta página será mostrada ao centro uma lista de programas disponíveis no modo *Trial* e *Free Version*. As duas primeiras opções (*CodeGear RAD Studio 2009* e *CodeGear RAD Studio 2009 ISO Disk 1 - Delphi and C++Builder*) são propriamente dito o instalador do Delphi 2009, com uma diferença. O primeiro link apenas baixa um instalador que faz toda a instalação e preparação do ambiente de desenvolvimento on-line. Já a segunda opção é recomendada para quem deseja ter a imagem de instalação do *RAD Studio*. Essa com certeza é a mais recomenda. Na página seguinte entre com as credenciais de acesso ao site e caso não as tenha, cadastre-se gratuitamente (**Figura 1)**.

Preencha o formulário corretamente para que seja lhe enviado por *e-mail* a chave de ativação do produto. Feito isso, clique no botão para fazer o *download* do arquivo de instalação. Esse processo requer que você informe o seu *login* do site da *CodeGear*. Caso não tenha, o cadastro ao *site* é gratuito.

Finalizado o *download*, descompacte o arquivo *installer.zip* e execute o *Install.exe*. A instalação será iniciada. Mantenha todas as opções padrão que lhe serão oferecidas. Informe também o código de ativação que você recebeu, quando este for solicitado. Note que durante esse processo, uma mensagem de alerta é mostrada, veja **Figura 2**.

Figura 1. Página de download e ativação da versão Trial do Delphi

Figura 2. Aviso sobre Ribbon Controls durante a instalação do Delphi 2009

Ela informa que você estará instalando o *Ribbon Controls* e que para distribuir aplicações desse tipo é necessário obter uma licença da *Microsoft* através do site informado. Clique em *OK* e dê prosseguimento até concluir a instalação. Em seguida abra o *Delphi 2009* e em sua *Tool Palette* repare a presença do novo grupo de componentes, **Figura 3**.

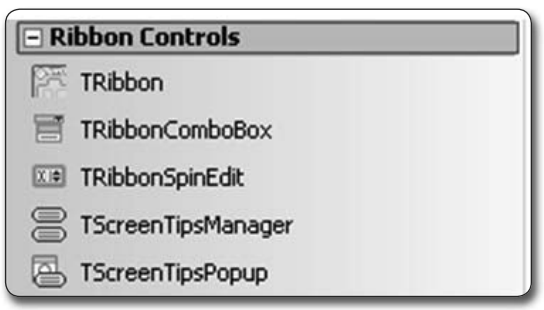


Figura 3. Paleta de componentes Ribbon

Iniciando a aplicação de exemplo

A suíte de componentes da paleta *Ribbon Controls* é bastante rica e possui os mais diversos controles. Não falarei sobre todos, apenas especialmente sobre os mais utilizados e importantes componentes da suíte. O primeiro deles e mais importante é o *TRibbon*. Entre os componentes da paleta, ele é o principal e funciona como um container, ou seja, dentro dele vão todos os demais componentes da aplicação. Para iniciar uma aplicação usando esses controles, crie uma nova aplicação *VCL* acessando o menu *File>New>VCL Forms Application – Delphi*. Salve tudo em uma pasta de sua preferência, nomeando os arquivos da seguinte forma: a *Unit* principal salve-a como *untEditorTexto.pas* e o nome do formulário como *frmEditorTexto*. Modifique o *Caption* da janela para *Editor de Textos – Ribbon*. Por fim salve o projeto com o nome *RibbonControls.dproj*.

No formulário adicione um *ActionManager*(*acmActionManager*) da paleta *Additional*, um *Ribbon* da paleta *Ribbon Contrrols*, dois *ImageList*(*imlImagesStd* e *imlImagesLarge*) da paleta *Win32*. No último adicione *Heigth* e *Width* para *32*. Isso fará com que as imagens fiquem maiores na tela. Por fim adicione um *RichEdit*(*redTexto*) também da paleta *Win32* e alinhado como *alClient*.

Ao adicionar o componente *Ribbon* perceba que por padrão ele assume a posição do topo do formulário, fazendo lembrarmos do aspecto visual do *Microsoft Office 2007*. Por fim, sua tela deverá estar parecida com a **Figura 4**.

Figura 4. Tela inicial do nosso exemplo com o componente Ribbon ao topo

Ligando os componentes e modificando o visual

Note que obedecendo esta seqüência, a propriedade *ActionManager* do *Ribbon* já fica devidamente vinculada ao componente que adicionamos anteriormente, o *acmActionManager*.

Agora precisamos ligar os componentes *ImageList* ao *ActionManager*. Primeiro, vincule o *imlImagesStd* à propriedade *Images*. Nele serão armazenadas as imagens padrão das ações. Em seguida, ligue o *imlImagesLarge* à propriedade *LargeImages*. Este conterá as imagens aumentadas das ações. Durante o desenvolvimento deste exemplo você perceberá a utilidade de cada um deles na prática.

Antes de começarmos ver as funcionalidades dos componentes *Ribbon Controls*, iremos fazer a última parte preparativa do nosso aplicativo que será adicionar as ações principais do nosso editor de textos no *ActionManager*. Ele possui várias ações já pré-configuradas, assim como o tradicional *ActionList*. Para adicioná-las, dê um duplo clique no componente e na tela que se abre, acione a opção *New Standard Action*. Mantendo pressionada a tecla *Ctrl* selecione os seguintes itens:

- *Edit: TEditCut, TEditCopy, TEditPaste, TEditUndo;*
- *Format: TRichEditBold, TRichEditItalic, TRichEditUnderline, TRichEditAlignLeft, TRichEditAlignRight, TRichEditAlignCenter;*
- *File: TFileOpen, TFileSaveAs, TFileExit;*
- *Dialog: TFontEdit.*

Clique em *OK* para confirmar e veja que agora o *ActionManager* mostra todas as ações adicionadas divididas por categoria e cada qual com sua imagem padrão associada. Essas imagens também já foram armazenadas automaticamente no *ImageList*. Confira, clicando duas vezes sobre o componente *imlImagesStd*. Utilizando essa mesma seqüência de imagens, *popule* o *imlImagesLarge* com figuras maiores, tamanho *32x32*. Só para nos mantermos dentro do contexto do artigo, nas ações *TFileOpen* e *TFileSaveAs*, configure a propriedade *Dialog>Filter* conforme a **Figura 5**, para que trabalhemos somente com arquivos textos.

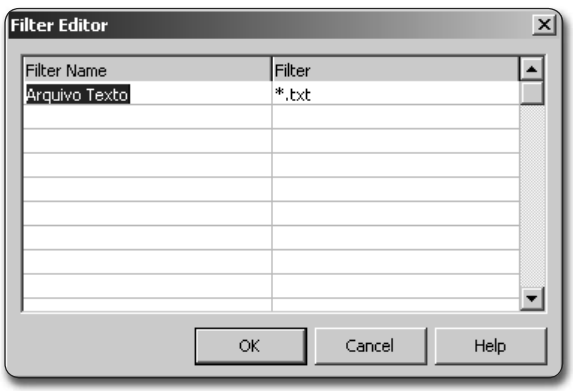


Figura 5. Adicionando um filtro para arquivos texto

Componente Ribbon

Agora veremos especialmente como trabalhar com o componente *Ribbon Control*. Veremos que sua utilização é bastante simples e não requer nenhuma ação especial para que ele funcione perfeitamente. Inicie clicando com o botão direito sobre ele. No menu suspenso que se abre, adicione duas *Tabs* através as opção *Add Tab*. Altere a propriedade *Caption* da primeira para *Editar* e o da segunda para *Outras Opções*. Cada *Tab* criada é uma *RibbonPage*. Observe que dessa forma podemos organizar melhor o menu de qualquer aplicação.

Ainda na primeira *Tab*, acrescente três *RibbonGroups*. Para isso, com a *Tab* selecionada, clique com o botão direito do *mouse* e escolha *Add Group*. Altere a propriedade *Caption* deles respectivamente para *Ações*, *Estilo* e *Alinhamento*. Nesses grupos iremos inserir as ações que configuramos anteriormente.

Para o primeiro grupo teremos as ações de *Cut*, *Copy*, *Paste* e *Undo* do *ActionManager*. Para adicioná-las dê um duplo clique sobre o *acmActionManager*, escolha a categoria *Edit* e arraste todos os seus itens para o *RibbonGroup Ações*. Perceba que

até aqui não escrevemos nenhuma linha de código, porém já temos um aplicativo funcional e com um visual moderno, veja a **Figura 6**.

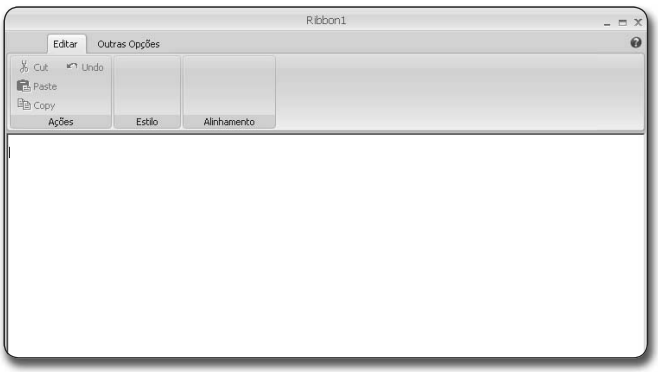


Figura 6. Aplicação de exemplo em execução

Como prova, execute a aplicação pressionando a *tecla F9* e faça alguns testes com as ações disponíveis até aqui.

Voltando ao desenvolvimento, iremos agora popular o grupo *Estilo*. Nele iremos abordar outro tipo de incremento visual do *Ribbon*, que é a possibilidade de alterarmos o estilo de exibição de cada item do grupo. Da mesma forma que fizemos anteriormente, arraste as ações *Bold, Italic* e *Underline*. Experimente selecionar o item *Bold* e em sua propriedade *CommandStyle*, mude para *csCheckBox*. Veja que o item agora virou um *CheckBox* e agora quando marcado, executará a ação relacionada. Repita este procedimento para os outros dois itens.

Aproveitando esse pensamento, vamos partir para o último *RibbonGroup* que falta, o *Alinhamento*. Este receberá as ações *Alignt Left, Align Right* e *Center*. O processo é o mesmo que o anterior, mas com um diferencial, ao selecionarmos os itens, mudaremos sua propriedade *CommandStyle* para *csRadioButton*. Intuitivamente pode-se perceber que tal mudança afetará seu comportamento de tal forma a trabalhar como um *RadioGroup* comum, onde sempre apenas uma única opção ficará selecionada dentre muitas. Execute novamente a aplicação e veja o funcionamento das novas implementações.

Incrementando o visual da aplicação

Tirando um pouco mais de proveito dos recursos visuais, vamos adicionar mais um *RibbonGroup* à *Tab Editar*, cujo *Caption* deverá ser *Fonte*. Ao selecioná-lo perceba que ele possui uma propriedade chamada *DialogAction* que contém uma lista de opções, que são justamente nossas ações do *ActionManager*. Para este caso selecione *FontEdit1*. Repare que no *RibbonGroup* apareceu um pequeno ícone no canto inferior direito, como na **Figura 7**.

Figura 7. Detalhe do RibbonGroup

Rode novamente a aplicação e veja que ao se clicar nesta opção, abre-se a janela de diálogo de fontes de texto do *Windows*, que foi a ação que associamos.

Voltando para a parte de desenvolvimento, apenas para mantermos nossa aplicação funcional, vamos codificar o evento *OnAccept* desta ação conforme código a seguir.

```
procedure TfrmEditorTexto.FontEdit1Accept(Sender: TObject);
begin
  //atribui-se à propriedade Font do RichEdit a fonte selecionada no
diálogo
  redTexto.Font:= FontEdit1.Dialog.Font;
end;
```

Dessa forma, quando o usuário clicar no *OK* da janela de diálogo, o texto do nosso editor será alterado também. Claro que essa não é a abordagem ideal, pois estaremos alterando o texto como um todo e não só a parte selecionada. Porém isto está fora do escopo deste artigo.

Outra alteração visual que podemos aplicar é mudar o tamanho do botão de uma ação dentro de um grupo. Para exemplificar, no *RibbonGroup Ações*, selecione a ação *Paste*. Altere suas propriedades *CommandProperties>ButtonSize* para *bsLarge* e *CommandProperties>ButtonType* para *bsSplit*. Mude também a opção *Items*, adicionando um item vinculado a uma *Action*. Para tal, adicionamos mais uma ação no *ActionManager* chamada *Paste Special*. Ao rodar a aplicação veja o efeito que esta funcionalidade nos traz, conforme **Figura 8**.

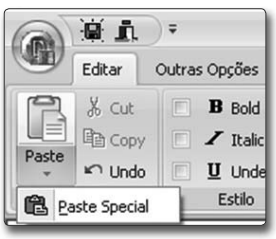


Figura 8. Botão da ação em formato de split, associado a uma outra ação

Note que o botão *Paste* agora está divido em duas partes. Na parte superior temos a ação em si, e na parte inferior temos um menu *dropdown* com uma outra ação vinculada.

Application Menu

Quem usa o *Microsoft Office 2007* sabe que um dos novos atrativos visuais dele é o ícone arredondado no canto superior esquerdo de seus programas, que substitui o antigo menu *Arquivo* das versões anteriores, veja **Figura 9**.

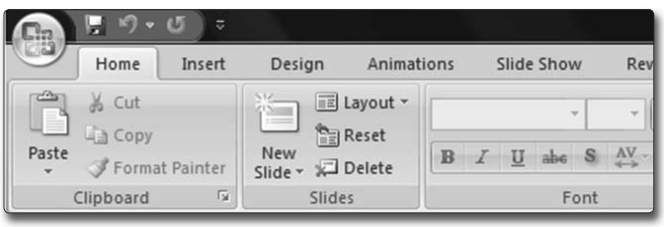


Figura 9. Detalhe do Application Menu no Word 2007

E é justamente esse elemento visual que iremos adicionar à nossa aplicação de exemplo. Selecione o componente *Ribbon*, clique com o botão direito do *mouse* e acesse seu menu suspenso escolhendo a opção *Add Application Menu*. Ao fazer isso surge no canto superior esquerdo um ícone grande. Ao ser selecionado, ele nos oferece um menu dividido em duas partes onde podemos adicionar funcionalidades ao sistema. É justamente isso que faremos agora. Do lado esquerdo acrescentaremos as ações e no lado direito listaremos os arquivos acessados recentemente pela aplicação.

Do *ActionManager* arraste para o *ApplicationMenu* as ações da categoria *File*, que são *Open*, *SaveAs* e *Exit*. O menu deverá ficar parecido com o da **Figura 10**.

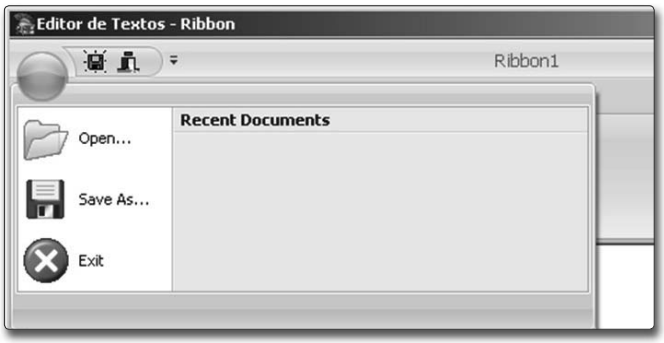


Figura 10. Ícones grandes sendo mostrados no Application Menu

Observe que os ícones grandes mostrados são aqueles que adicionamos no *ImageList imlImagesLarge*. Então concluímos que no *ActionManager* propriedade *Images* contém as imagens usadas por *RibbonCommands*, itens comuns. Enquanto que a propriedade *LargeImages* é usada por *Ribbon Application Menu* e *Large Buttons*. Este último será visto mais adiante.

Mantendo a idéia de que apesar de simples é fundamental deixarmos nossa aplicação funcional, vamos codificar mais um pouco. Nos eventos *OnAccept* das ações *Open* e *SaveAs*, insira os comandos conforme a **Listagem 1**.

Observe que na linha 7 estamos carregando o arquivo previamente selecionado. O nome desse arquivo foi armazenado na variável *NomeArquivo* e na linha 8 passamos esse nome para propriedade *DocumentName* do *Ribbon*. Veja também como é simples manter e exibir a lista dos últimos arquivos utilizados. O *Ribbon* oferece o método *AddRecentItem* que adiciona o nome do arquivo à essa lista. De forma semelhante fazendo no método *FileSaveAs1Accept*.

Listagem 1. Configurando as ações para manipulação de arquivos

```
01 procedure TfrmEditorTexto.FileOpen1Accept(Sender: TObject);
02 var
03   NomeArquivo: string;
04 begin
05   redTexto.Lines.Clear;
06   NomeArquivo:= FileOpen1.Dialog.FileName
07   redTexto.Lines.LoadFromFile(NomeArquivo);
08   Ribbon1.DocumentName:= NomeArquivo;
09   Ribbon1.AddRecentItem(NomeArquivo);
10 end;
11 procedure TfrmEditorTexto.FileSaveAs1Accept(Sender: TObject);
12 var
13   NomeArquivo: string;
14 begin
15   NomeArquivo:= FileSaveAs1.Dialog.FileName;
16   redTexto.Lines.SaveToFile(NomeArquivo);
17   Ribbon1.DocumentName:= NomeArquivo;
18   Ribbon1.AddRecentItem(NomeArquivo);
19 end;
```

Outro evento que precisaremos codificar é o *OnRecentItemClick* do componente *Ribbon*, para que no momento em que o usuário clicar sobre algum item da lista de arquivos recentes, este seja imediatamente aberto, funcionando como uma espécie de *hiperlink*. Para isto, digite o código da **Listagem 2**.

Listagem 2. Abrindo um arquivo da lista de recentes

```
01 procedure TfrmEditorTexto.Ribbon1RecentItemClick(Sender: TObject;
02   FileName: string; Index: Integer);
03 begin
04   redTexto.Lines.Clear;
05   redTexto.Lines.LoadFromFile(FileName);
06   Ribbon1.DocumentName:= FileName;
07 end;
```

Repare que estamos atribuindo o nome do arquivo à propriedade *DocumentName* do *Ribbon*. Esse é o texto que aparece na parte superior da aplicação. Mais uma vez, vamos testar nosso exemplo para vermos o funcionamento dessas novas funcionalidades. Então execute a aplicação.

Acessando o *ApplicationMenu* abra alguns documentos textos, salve-os com algum nome diferente e vá observando a lista de arquivos recentes do lado direito sendo populada. Tente também clicar sobre esses itens e veja os arquivos sendo carregados para o nosso editor.

Quick Access Toolbar

Outro elemento gráfico presente no *Ribbon* é o *Quick Access Toolbar*. Novamente citando o *Microsoft Office 2007*, especificamente o *Word*, lembramos que ao lado do ícone *Application Menu* existem alguns botões pequenos de acesso rápido, como na **Figura 11**. Esses botões são os que estaremos acrescentando agora em nosso exemplo.

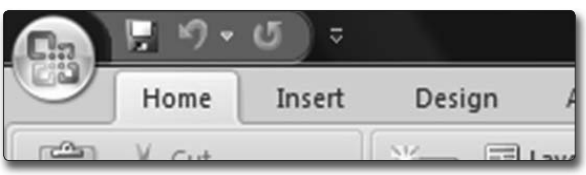


Figura 11. Botões de acesso rápido, dispostas em uma Quick Access Toolbar

Primeiramente dê um clique com o botão direito do *mouse* no componente *Ribbon* e acesse a opção *Quick Access Toolbar* para que ele fique disponível. Em seguida adicionaremos as ações de acesso rápido que queremos que fiquem à mostra naquela área do programa. Arraste do *ActionManager* as ações *SaveAs* e *Exit*. Note que além dessas opções também é habilitado um outro ícone ao lado dos botões. Ao executar a aplicação clique sobre ele e veja que é aberto um menu *dropdown* como na **Figura 12**.

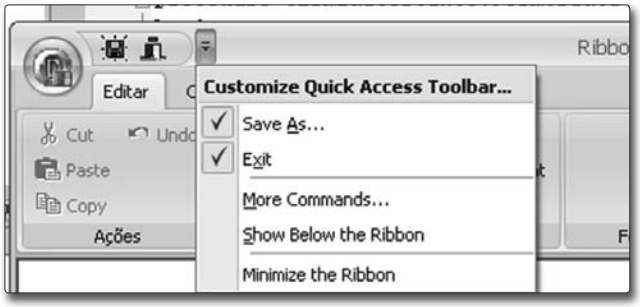


Figura 12. Menu drop down no Quick Access Toolbar

Esse menu *dropdown* contém as seguintes opções:
• *SaveAs*: ação que foi associada ao *Quick Access Toolbar*;
• *Exit*: ação que foi associada ao *Quick Access Toolbar*;
• *More Commands*: abre a tela de customização do *ActionManager*, onde o usuário poderá selecionar qualquer outra ação e acrescentá-la à *Quick Access Toolbar*, utilizando o movimento de arrastar e soltar;
• *Show Below the Ribbon*: posiciona a toolbar abaixo do componente *Ribbon*;
• *Minimize the Ribbon*: minimiza o componente *Ribbon*.

Help Button

Por padrão, quando adicionamos um componente *Ribbon* ao formulário, a sua propriedade *ShowHelpButton* vem assinalada como *True*. Essa opção habilita um botão na parte direita do componente na forma de círculo com o ícone de interrogação. Obviamente este é o local exato para que seja implementado, ou chamado, um mecanismo de ajuda do sistema. Apesar de não fazer parte do objetivo do artigo, mas para que não passe em branco, vamos codificar este botão com uma mensagem simples através do seu evento *OnHelpButtonClick*, conforme código a seguir. Veja que é simples a implementação, ou seja, se tivéssemos um arquivo de *Help* que pudéssemos associar ao aplicativo, bastaria chamá-lo.

```
procedure TfrmEditorTexto.Ribbon1HelpButtonClick(Sender:
TObject);
begin
  Application.MessageBox('Sistema de ajuda - Editor de Texto
Ribbon', 'Ajuda', MB_ICONINFORMATION);
end;
```

RibbonComboBox e RibbonSpinEdit

Veremos agora outros dois componentes da paleta *Ribbon Controls*. Arraste para dentro do grupo *Fonte* um *RibbonComboBox* e um *RibbonSpinEdit*. Neste momento faremos algo diferente para que você perceba que não é só através das *actions* que podemos adicionar funcionalidades ao *Ribbon*. Para o *combo*, iremos populá-lo com as fontes de texto do *Windows* e quando o usuário selecionar um de seus itens, o efeito será o mesmo de quando usamos a ação *TFontEdit*. Configure-o também como somente leitura, mudando sua propriedade *ReadOnly* para *True*. A seguir, codifique os evento *OnCreate* do formulário e *OnChange* do *RibbonComboBox* conforme a **Listagem 3**.

Listagem 3. Manipulando os eventos relacionados ao ComboBox

```
01 procedure TfrmEditorTexto.FormCreate(Sender: TObject);
02 begin
03   RibbonComboBox1.Items:= Screen.Fonts;
04 end;
05
06 procedure TfrmEditorTexto.RibbonComboBox1Change(Sender:
    TObject);
07 begin
08   redTexto.Font.Name:= RibbonComboBox1.Text;
09 end;
```

Na linha 3 estamos carregando os nomes das fontes instaladas para o *RibbonComboBox1* e na linha 9 aplicamos a fonte selecionada ao texto. Já com o *RibbonSpinEdit* implementaremos o tamanho das fontes de texto. Para deixarmos nosso exemplo amigavelmente funcional, altere as seguintes propriedades para:
• *MinValue* = 8;
• MaxValue = 72;
• *Value* = 8.

E codifique seu método *OnChange* do *RibbonSpinEdit* conforme o seguinte código:

```
procedure TfrmEditorTexto.RibbonSpinEdit1Change(Sender: TObject);
begin
  //configura-se a altura das fontes, conforme o valor do SpinEdit
  redTexto.Font.Height:= RibbonSpinEdit1.Value;
end;
```

Veja que estamos aplicando o tamanho atual contido no *RibbonSpinEdit1*. Note também que ao adicionarmos esses dois componentes visuais dentro de um *RibbonGroup*, eles automaticamente ficaram associados a um *TActionClientItem*, que é o item de uma *ActionBar* do *ActionManager*. Portanto, para alterar o texto do *Label* que acompanha os controles, é necessário selecionar a *TActionClientItem* relacionada. Para selecioná-la você pode clicar sobre o *Label* do objeto ou ir pela estrutura *treeview* do *ActionManager*, conforme a **Figura 13**.

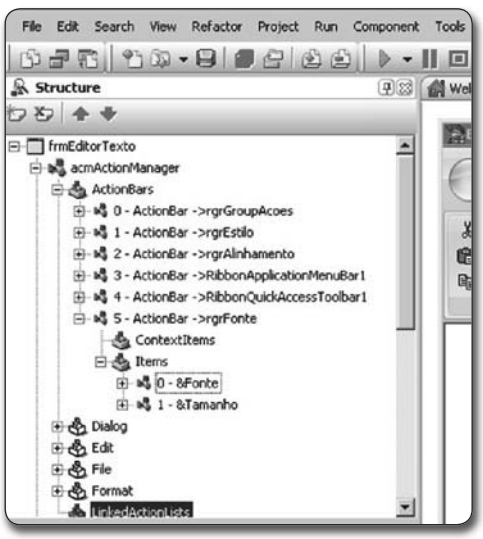


Figura 13. Estrutura treeview do ActionManager

Com a *TActionClientItem* do *ComboBox* selecionada, altere seu *Caption* para *Fonte*. Expanda sua propriedade *CommandProperties* e veja que nela é apresentado o controle associado, bem como seu tamanho e o tamanho do *Label*. Faça o mesmo processo para o *Label* do *SpinEdit*.

Rode a aplicação e faça alguns testes.

Key Tips

Key Tip é uma propriedade presente nos componentes da família *Ribbon Controls* e nas ações do *ActionManager*. Ela recebe como valor uma ou mais letras, todas maiúsculas, ou números, que representam justamente a tecla a ser pressionada para acessar tal elemento visual, seja ele uma *Tab*, um grupo ou uma ação. Uma espécie de tecla de atalho. Vamos agora definir uma *Key Tip* para cada elemento *Ribbon* e suas ações, como por exemplo:

- RibbonPage1: KeyTip = 'E';
- RibbonPage2: KeyTip = 'O';

Feito isso execute a aplicação. Você vai reparar que mesmo apertando a tecla atribuída à propriedade *Key Tip*, nada acontece. Isso porque, para fazer uso deste recurso, primeiro o usuário deve pressionar e segurar a tecla *Alt*. Será mostrado então sobre cada elemento, uma dica na forma de um pequeno balão contendo sua tecla de acesso, como pode ser visto na **Figura 14**.

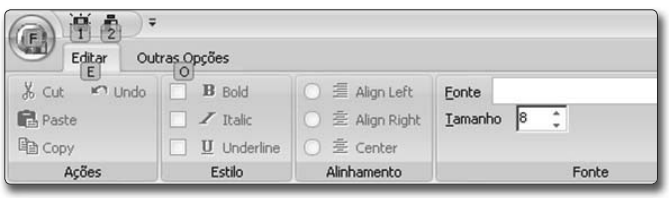


Figura 14. Key tips em exibição no nosso exemplo

A partir daí o usuário poderá pressionar a *Key Tip* relacionada ao elemento que quiser. Rode a aplicação novamente e experimente navegar entra as duas *Tabs* de nosso editor de textos.

Screen Tips Manager

Na paleta *Ribbon Controls* existe ainda um componente chamado *ScreenTipsManager* que nos permite fazer alguma personalização, uma espécie de *hint* sofisticado relacionadas à cada ação do *ActionManager*.

Adicione um *ScreenTipsManager* ao formulário. Em sua propriedade *LinkedActionList* vincule o nosso componente *ActionManager*. Depois, clique com o botão direito do mouse para acessar seu menu de opções. Acione então a opção *Generate Screen Tips*. Isso fará com que todas as ações do *ActionManager* sejam vinculadas ao *ScreenTipsManager*. Por fim, faça uma outra ligação, agora com o componente *Ribbon* através da propriedade *ScreenTips* do mesmo. Altere também o *ShowHints* do formulário para *True*. Já podemos testar o efeito visual disso. Rode a aplicação e perceba que no passar do mouse sobre cada ação é exibido um *hint* mais atraente, em forma de balão, como pode ser visto na **Figura 15**.

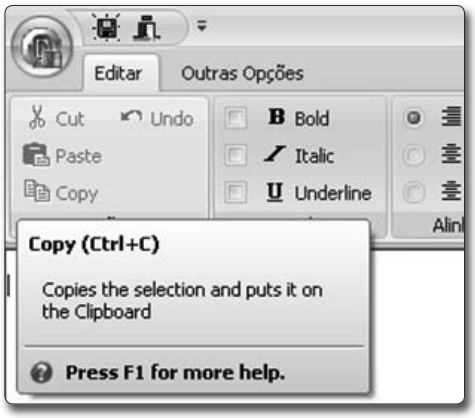


Figura 15. Screen tip sendo mostrado

Mas ainda falta o principal, que é a possibilidade de personalizarmos esses balões de diálogo. Para isso, clique duas vezes sobre o componente *ScreenTipsManager*. Será aberto um editor para personalização da *tip* de cada ação. O interessante é que é possível testar o resultado visual neste mesmo instante através do botão *Preview*. Veja **Figura 16.**

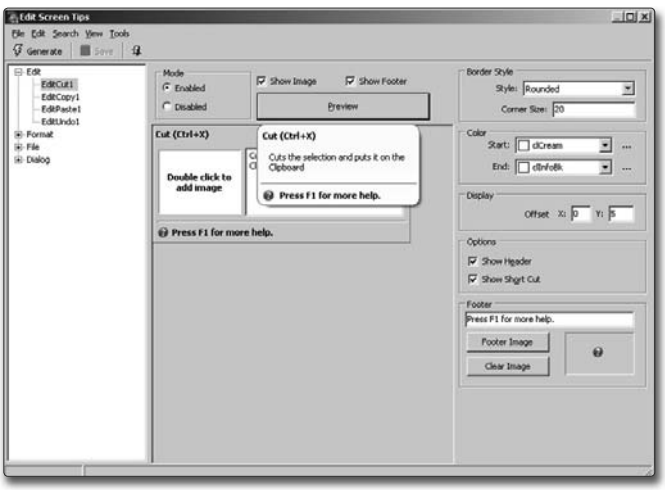


Figura 16. Editor de screen tips

Screen Tips Popup

O último componente da *Ribbon Controls* é o *ScreenTipsPopup*. Ele é um item visual que funciona como uma espécie de *Screen Tip* individual. Para usá-lo você deverá fazer três vínculos importantes através das seguintes propriedades:

- *Associate* – associe-o a algum elemento da família *Ribbon*;
- *ScreenTipManager* – aponte para o componente *ScreenTipsManager* de nosso projeto;
- *ScreenTip>Action* – vincule-o a uma ação.

Ainda na propriedade *ScreenTip* é necessário fazer algumas observações. Note que ao expandí-la, são mostrados os seguintes itens:

- *Description* – será o texto do *ScreenTip*, quando a ação que você vinculou estiver habilitada;
- *DisableDescription* - será o texto do *ScreenTip*, quando a ação que você vinculou estiver desabilitada;

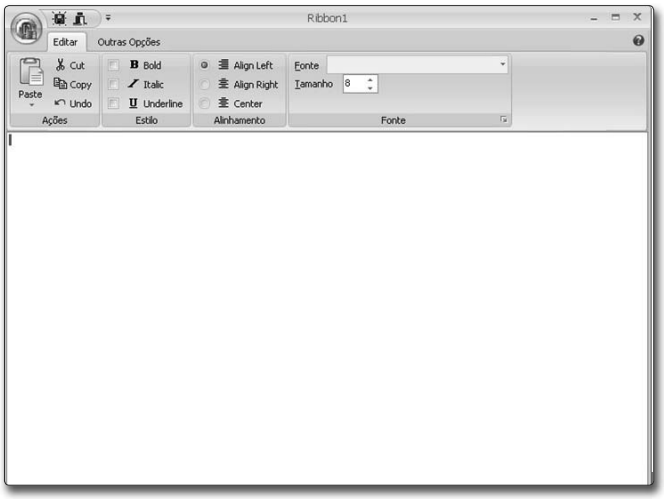


Figura 17. Resultado final de nossa aplicação teste

• *DisableHeader* – será o cabeçalho do *ScreenTip*, quando a ação que você vinculou estiver desabilitada;
• *Header* - será o cabeçalho do *ScreenTip*, quando a ação que você vinculou estiver habilitada;

Como teste, insira tal componente, configurando-o conforme as explicações anteriores e execute a aplicação. Perceba que ao passar o mouse sobre a ação que você fez o vínculo, já não é mais mostrada sua *ScreenTip*, que agora será visualizada no passar do *mouse* sobre o ícone do *ScreenTipsPopup* que adicionamos ou sobre o componente da família *Ribbon* que associamos. Teste também o seu comportamento com a ação habilitada e desabilitada.

Confira o resultado final de nosso exemplo na **Figura 17**.

Conclusão

Neste artigo, vimos um pouco do que pode ser feito com os novos componentes *Ribbon* que vieram no *Delphi 2009*. Com certeza nós desenvolvedores ganhamos mais um leque de recursos para estarmos agregando valor aos nossos aplicativos, principalmente na questão de experiência do usuário. Imagine uma aplicação comercial desenvolvida com este novo padrão visual. Certamente seu sistema traria um diferencial agradável para o cliente.

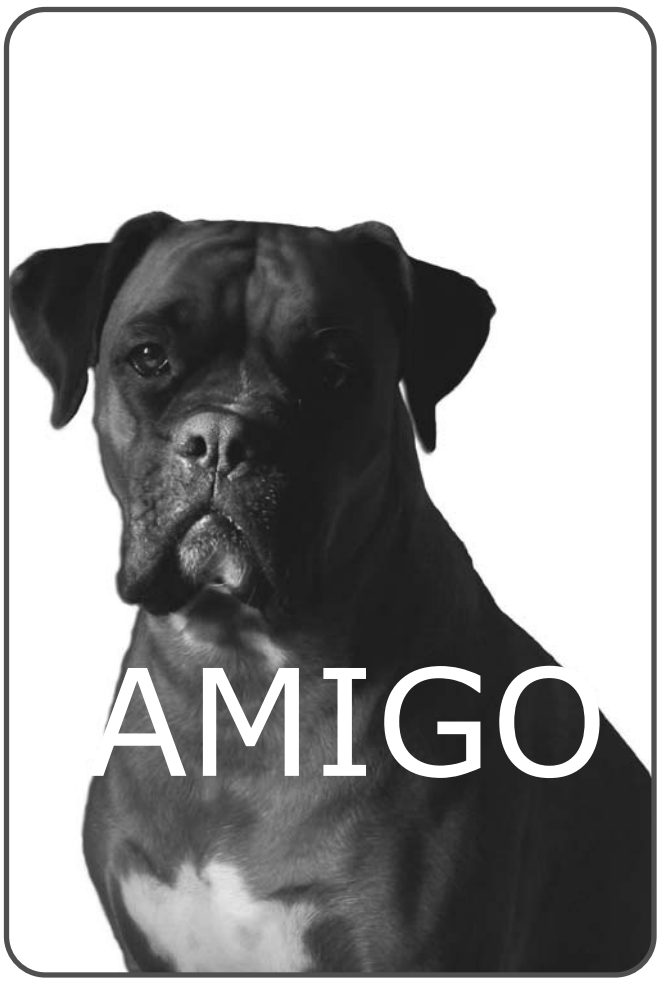
Espero ter ajudado a todos. Até a próxima, obrigado. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback



Existem coisas que não conseguimos ficar sem!

...só pra lembrar, sua assinatura pode estar acabando!

Renove Já!

www.devmedia.com.br/renovacao

Para mais informações:
www.devmedia.com.br/central

DevMedia GROUP

Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

Delphi for PHP x Ajax

Utilize Ajax em aplicações Delphi for PHP 2.0

Resumo do DevMan

Ajax já não é mais segredo para ninguém, ele está ai e todo mundo conhece. De fato o Ajax veio para ficar e contribuir ainda mais para agilizar os web sites na internet. Com ele a vida do internauta fica mais confortável. Não há hoje nenhuma contra-indicação para aplicação de Ajax. Toda e qualquer página pode fazer uso da tecnologia.

Nesse artigo veremos como aplicar essa tecnologia em aplicações desenvolvidas em Delphi for PHP 2.0, o que não é difícil haja vista que o D4PHP já tem recursos suficientes para suportar o Ajax.

Nesse artigo veremos
Como utilizar Ajax em aplicações Web com Delphi for PHP;

Qual a finalidade
Agilizar o processo de atualização da página evitando postback's o tempo todo;

Quais situações utilizam esse recurso?
Em qualquer tipo de aplicação web é importante aumentar a velocidade de carregamento das páginas, portanto utilizar Ajax é importantíssimo em qualquer sitação;

Adriano Santos
falecom@adrianosantos.pro.br
É desenvolvedor Delphi desde 1998, professor e programador PHP, bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É Editor Chefe das revistas ClubeDelphi e WebMobile. Gerente de TI na SoftPark, parceira Borland, membro fundador do DUG-SP (www.dug-sp.com) e mantenedor do blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre o desenvolvimento Delphi.

Quem nunca se irritou com lentidão de uma conexão com a internet em casa ou no trabalho que levante a mão. E quem nunca abriu um site de concurso para fazer uma inscrição e descobriu que a cada campo que preenche a tela fica branca e demora *pacas* para carregar novamente? De fato a internet hoje faz parte de nossa vida há tempos, mas conviver com esses problemas é que deixa a desejar muitas vezes. Eu costumo dizer que o site do concorrente está a um *click* daqui, portanto se seu site não objetivo, atraente e, principalmente, rápido, seu cliente vai embora e sabe-se lá quando volta. Não dá para pegar o internauta pelo braço e dizer: tome mais um café.

Agilidade hoje em dia é sinônimo de boas vendas na internet, as vezes até mais

do que segurança. Mas como fazer para que nosso *web* site seja rápido e atraente? Bem, atraente fica por conta do *webdesigner*, é função dele fazer isso, mas rápido, nós mesmos podemos tomar essa providência. Quem dera se tivéssemos *Ajax* na época dos modens *U.S Robotics de 56 Kbps*. A solução para nossos problemas de velocidade é *Ajax*, sem dúvida nenhuma.

Vamos fazer uma pequena aplicação nesse artigo utilizando essa tecnologia com *Delphi for PHP* para vermos como isso é simples e rápido de ser aplicado na prática.

Entendendo o Ajax

O que exatamente é Ajax? Ajax na verdade não é nenhuma nova tecnologia, assim como já expliquei em outro artigo meu, mas dessa vez com *VCL for the Web*. Na realidade o Ajax é um conjunto de tecnologias já existentes e que unidas formam um recurso bastante interessante de ser aplicado em *web sites*. Na internet encontramos milhares de definições para o termo. Mas confesso que muitas delas acabam fazendo o leitor achar que Ajax é algum time de futebol ou mesmo um tipo de desinfetante ou produto de limpeza. Vejamos de fato o que é Ajax.

Como já comentei, ele é um conjunto de tecnologias que já existiam e quem unidas deixam sua aplicação mais dinâmica evitando inclusive o *postback*, o recarregamento da página. Significa *Asynchronous JavaScript And XML*. No site *www.yourhtmlsource.com/javascript/ajax.html* você vai encontrar esta definição em inglês, além de muito mais a respeito do assunto. Encontrei na internet as definições a seguir e achei de bom tamanho e muito interessante publicar para que o leitor possa entender de uma vez por todas o que é *Ajax*.

• *Assíncrono:* Significa que quando você faz uma requisição e espera o retorno, mas está livre para fazer outras coisas enquanto isso. A resposta provavelmente chega quase que imediatamente, então você faz uma função que espera o envio do servidor e faz alguma coisa quando ela chegar, ou seja, assíncrono quer dizer *pede para o servidor, você faz o que tiver que fazer e quando ficar pronta, ele te manda a resposta*;

• *Javascript:* É uma linguagem para criar *scripts* e deixar suas páginas mais dinâmica. Neste caso, do *Ajax*, usado para requisitar ao servidor. Uma vez que a resposta é retornada do servidor, você normalmente vai usar alguma(s) função(ões) *Javascript* para mudar algo na página atual para mostrar que a ação foi realizada com sucesso;

• XML: É uma linguagem de marcação que armazena somente dados. Os dados que você recebe do servidor são empacotados como XML, sendo facilmente processado pelo Javascript. Este dado pode qualquer coisa de qualquer tamanho.

Como podemos ver, Ajax não é tão novo assim. Mas isso não importa muito, o que é mais importante aqui é que o seu uso melhora a usabilidade, agiliza o carregamento da página, e abre um leque de opções para aumentar ainda mais o dinamismo do web site e interação do usuário com ele. Ajax faz parte da definição de Web 2.0 que tanto se fala nos dias de hoje. Portanto, aplicar essa tecnologia hoje é pensar em aplicações mais robustas e preparadas para o dia-a-dia.

Nota do DevMan

Em poucas palavras, XML (eXtensible Modeling Language) pode ser definida como uma linguagem de marcação de dados extensível - ao contrário do HTML - que foi projetada para permitir o uso do SGML (Standard Generalized Markup Language) na World Wide Web. Ela provê um formato para descrever dados estruturados que facilita declarações mais precisas do conteúdo.

O XML não é uma simples linguagem de marcação pré-definida: Ela é uma metalinguagem - uma linguagem usada para descrever outras linguagens - que permite que o seu usuário defina a sua própria marcação. Uma linguagem de marcação pré-definida como o HTML especifica um modo de descrever informação em apenas uma classe específica de documento. O XML permite que o usuário defina as suas próprias linguagens de marcação para atender à inúmeras classes de documentos diferentes. Isto é possível porque o XML é escrito em SGML, a metalinguagem padronizada internacionalmente para sistemas de marcação de texto.

Nota do DevMan

JavaScript é uma linguagem de programação criada pela Netscape em 1995, que a princípio se chamava LiveScript, para atender, principalmente, as seguintes necessidades:
* Validação de formulários no lado cliente (programa navegador);
* Interação com a página. Assim, foi feita como uma linguagem de script. Javascript tem sintaxe semelhante a do Java, mas é totalmente diferente no conceito e no uso;
* Oferece tipagem dinâmica - tipos de variáveis não são definidos;
* É interpretada, ao invés de compilada;
* Possui ótimas ferramentas padrão para listagens (como as linguagens de script, de modo geral);
* Oferece bom suporte a expressões regulares (característica também comum a linguagens de script).;
* Sua união com o CSS é conhecida como DHTML. Usando o Javascript, é possível modificar dinamicamente os estilos dos elementos da página em HTML;

Dada sua enorme versatilidade e utilidade ao lidar com ambientes em árvore (como um documento HTML), foi criado a partir desta linguagem um padrão ECMA, o ECMA-262, também conhecido como ECMAScript. Este padrão é seguido, por exemplo, pela linguagem ActionScript da Macromedia.

Além de uso em navegadores processando páginas HTML dinâmicas, o JavaScript é hoje usado também na construção do navegador Mozilla, o qual oferece para a criação de sistemas GUI todo um conjunto de ferramentas (em sua versão normal como navegador, sem a necessidade de nenhum software adicional), que incluem (e não apenas) um interpretador de Javascript, um comunicador Javascript <-> C++ e um interpretador de XUL, linguagem criada para definir a interface gráfica de aplicações.

O uso de JavaScript em páginas XHTML, pelo padrão W3C, deve ser informado ao navegador da seguinte forma: escreve na página

```
<script type="text/javascript">
  /* aqui fica o script */
</script>
```

Caso contrário, o navegador irá interpretar o script como sendo código HTML, escrevendo-o na página.

Nota do DevMan

CSS significa, Cascade Style Sheet, ou Folha de Estilos em Cascata. Uma especificação em HTML desenvolvida pelo World Wide Web Consortium, que permite aos autores de páginas da Web anexarem folhas de estilos em documentos HTML.

As folhas de estilos podem incluir informações tipográficas sobre como a página deve aparecer, como a fonte do texto na página. A CSS também controla a maneira como as folhas de estilo do documento HTML e o estilo do usuário serão combinados.

Criando o banco de dados

A idéia desse artigo é criar uma aplicação onde carregaremos os pedidos de um determinado cliente. Teremos um *ComboBox* com os nomes de clientes e ao trocar de cliente um *DBGrid* mostrará os dados na tela, simples assim. Utilizaremos um banco de dados *MySQL Alpha 6.09* e o *Delphi for PHP 2.0*. De qualquer forma é perfeitamente possível trabalhar também com outra versão ou mesmo outro banco de dados, como, *Firebird e Interbase*, entre outros. O *Delphi for PHP 1.0* também contempla esses recursos apresentados aqui.

O *MySQL* pode ser encontrado no site www.mysql.com, ou ainda *www.superdownloads.com.br*. Não entrarei em detalhes quanto a instalação de ambas ferramentas por não fazer parte do escopo desse artigo.

Outra ferramenta que farei uso é o *HeidSQL*, antigo *MySQL Front*. Com ele é possível ter acesso ao *MySQL*, criar nossas bases de dados, tabelas e tudo que diz respeito ao banco de dados. A ferramenta é *freeware* e pode ser encontrada no site *www.heidisql.com*. Baixe a versão e faça sua instalação que não requer nenhuma configuração especial.

Após criado o ambiente de desenvolvimento, abra o *HeidSQL* ou seu *front-end* para *MySQL* que preferir, e crie um banco de dados com o nome *Ajax*. Para quem tem o *HeidSQL* instalado, a partir de agora faremos a criação passo-a-passo do banco e das tabelas necessárias. Clique com o botão direito no hostname do HeidSQL e escolha *Create Database*, ou se preferir clique no atalho na barra de ferramentas superior. Na janela que se abre (**Figura 1**) digite o nome do banco e dados (*Ajax*) e confirme em Ok.

Com o banco de dados criado, note que a esquerda do programa é mostrada uma árvore, portanto expanda o item *Ajax* (nosso banco de dados) e vamos dar início à criação das tabelas do banco. Clique com o botão direito do mouse sobre o *BD* e escolha *Create a new table*. Dê o nome de *Clientes* e em seguida inicie a inclusão de campos.

A esquerda da janela você encontrada um *Edit* com o texto *Enter column name*. Apague o texto e digite o nome do campo (*ID*) clicando depois em *Add*. Todas as propriedades do campo encontram-se a direita da tela. Para esse campo defina que ele será do tipo *INT*(*Type*), *Primary, Unique, Not Null* e *AutoIncremer*. Em seguida volte ao *Edit* e digite o nome do segundo campo clicando na seqüência em *Add*. Esse campo será um *VarChar(50)*. Se preferir insira novos campos tais

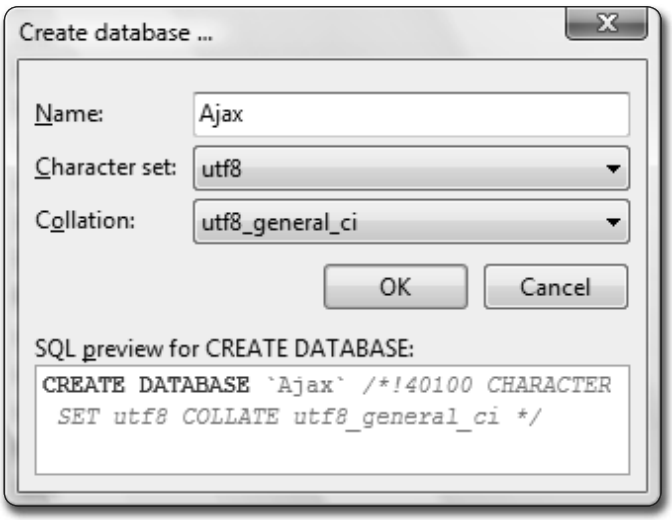


Figura 1. Criando o banco de dados

como *Razão, CNPJ*, etc. A princípio somente esses campos já são suficientes para que possamos fazer nosso exemplo funcionar. Para finalizar a criação da tabela, clique em *Create* (**Figura 2**).

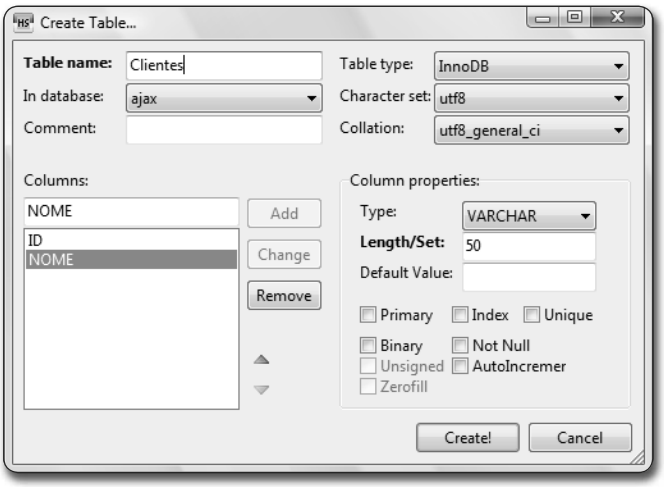


Figura 2. Criação da tabela de clientes

Repita o processo e crie uma segunda tabela chamada Pedidos. Ela deverá conter os campos *ID, ID_CLIENTE, NUM_PEDIDO, VALOR e PRODUTO*. A definição completa de todos os campos poder ser vista na **Tabela 1**.

Repare que criamos um campo que fará referência a tabela clientes, ou seja, conseguiremos saber todos os pedidos que o cliente *X* fez. É interessante criar uma chave estrangeira (*Foreign Key*) para *linkar* ambas tabelas, mas somente dessa forma já é possível fazer nossos testes, por isso não entrarei em detalhes.

Feito isso, clique duas vezes em cada tabela e inclua registros em ambas, para que nossos testes sejam bem sucedidos. No meu caso minhas tabelas ficaram semelhantes as **Figuras 3**. Para termos certeza de que os dados estão corretos, acesse a aba *Query* e digite uma instrução SQL como a seguir:

Campo	Tipo e Tamanho	Especial
ID	INT	Primary, Unique, Not Null e AutoIncremer
ID_CLIENTE	INT	
NUM_PEDIDO	INT	
VALOR	FLOAT	
PRODUTO	VARCHAR(50)	

```
SELECT * FROM
  CLIENTES CL
  INNER JOIN PEDIDOS PD
  ON (PD.ID_CLIENTE = CL.ID)
```

O resultado é o que podemos conferir na **Figura 4**. Veja que cada cliente tem o seu produto listado no resultado. O que faremos no sistema é listar nossos cliente em um *ComboBox* e então mostrar seus pedidos conforme o *Combo* é alterado, ou seja, o cliente é escolhido. Isso nós faremos utilizando Ajax, para evitar que a página fique branca toda vez que filtrar os pedidos do cliente escolhido.

Criando a aplicação

Muito bom, até agora temos o banco criado e um ambiente de desenvolvimento preparado para criar nossa aplicação. O próximo passo agora é criar a aplicação. Para isso abra o *Delphi for PHP 2.0* e vamos começar a brincadeira. Você verá que não há segredos na utilização de Ajax e que não é nenhum bicho de 7 cabeça fazer com a aplicação fique rápida, profissional e robusta. Mãos na massa.

Inicie criando uma aplicação usando o menu *File>New>Application*. Em seguida crie um diretório na unidade que desejar e salve a *Unit* principal como *index.php* e o projeto como *Ajax.phprj*. Na página principal vamos incluir os componentes de acesso a dados, mas antes vamos registrar o MySQL para que tenhamos acesso a ele. Acesse o *Data Explorer* a direita do *D4PHP* e clique com o botão direito do mouse em *MySQL*, acesso *Register DataBase*. Preencha os dados de conexão com o banco de dados. *Connection Name* é apenas o nome da conexão que aparecerá no *Data Explorer*. Preencha o *Host* como *localhost*, usuário igual a *root* e a senha que definiu na instalação do *BD*. Se preferir inclua também no nome do *BD*, *Ajax* no nosso caso (**Figura 5**). Se tudo correr bem, basta agora conectar-se ao banco de dados via *Data Explorer*. Um cenário semelhante ao **Figura 6** é esperado.

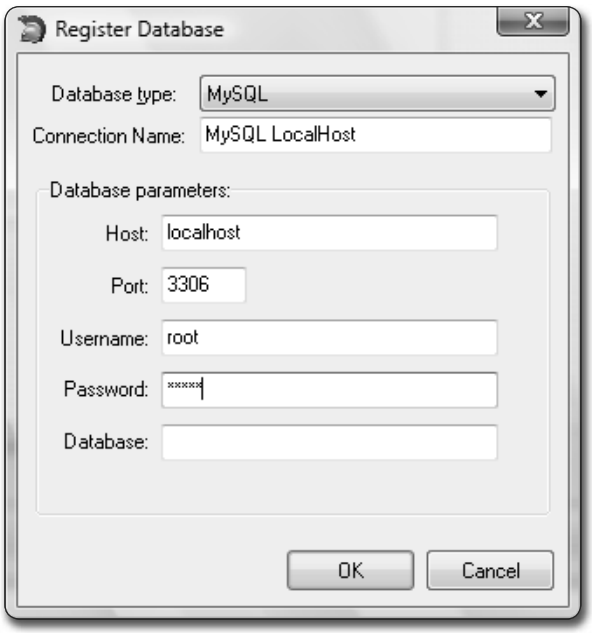


Figura 5. Dados de conexão com o banco

Para certificar-se de que tudo está certo, arraste para o *index.php* a tabela *Clientes*. Serão adicionados um *DBGrid*, um *DataBase*, um *Table* e um *DataSource*, todos devidamente configurados e prontos para funcionamento. Organize como desejar e em seguida execute a aplicação pressionando *F9*. As visões em modo de projeto e runtime, respectivamente, podem ser vistas nas **Figuras 7** e **8**.

Sem Ajax como o site se comporta?

Para que entendamos corretamente o funcionamento, vamos fazer uma pequena alteração no nosso sistema para que ele comece a tomar forma. Inclua um *ComboBox*(*cbxClientesGeral*) na tela e acima dele um *Label* com o *Caption* alterado para *Clientes*. O *ComboBox* receberá duas informações, sendo a primeira o nome do Cliente em sua propriedade *Items* e o *ID* dele em sua propriedade *Value*. Para isso faremos um loop na tabela de clientes adicionado esses dados no *Combo*.

Selecione o formulário e acesse seu evento *OnCreate*. Nele digite o código da **Listagem 1**. Repare que o código é bem simples e semelhante ao que fazemos com aplicações *Win32*. Abrimos a tabela, movimentamos o ponteiro para o primeiro registro,

ID	NOME	ID_1	ID_CLIENTE	NUM_PEDIDO	VALOR	PRODUTO
1	Adriano Santos	1	1	123	100	Livro: A vida de um cão
1	Adriano Santos	2	1	890	200	Livro: SQL Server a Bíblia
3	Gladstone Matos	3	3	1125	120,1	Livro: Saúde em primeiro lugar
4	Valéria de Pádua	4	4	1313	5,25	Livro: Deus na terra e eu no céu

Figura 4. Resultado da instrução SQL com Join das tabelas

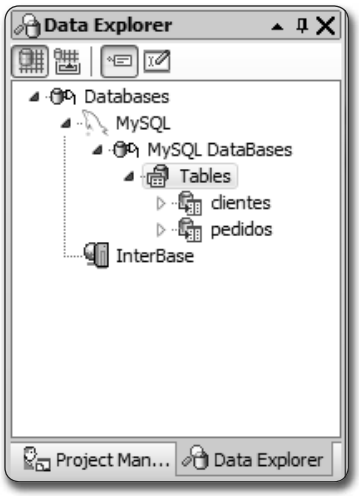


Figura 6. Data Explorer conectado ao MySQL

fazemos um laço *while* na tabela e adicionamos os dados na propriedade *Items* do *ComboBox*. O primeiro parâmetro do método *AddItem* adiciona o valor que aparecerá no texto do item do *combo*, ou seja, o que o usuário verá. O segundo é o ponteiro para algum objeto em tela, que no nosso caso estamos passando como nulo. E por último informamos qual o ID do cliente adicionado. Esse valor é acrescentado na propriedade *Value* do item.

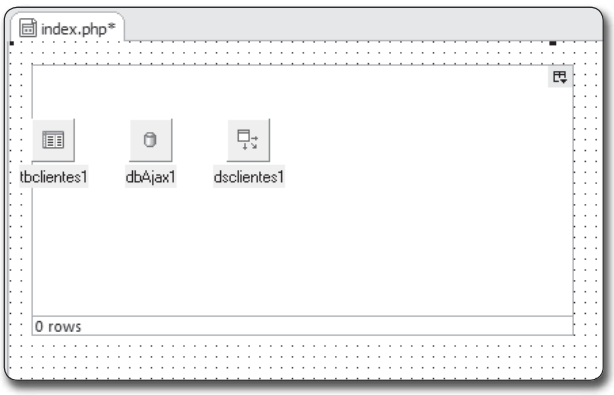


Figura 7. Site em tempo de projeto

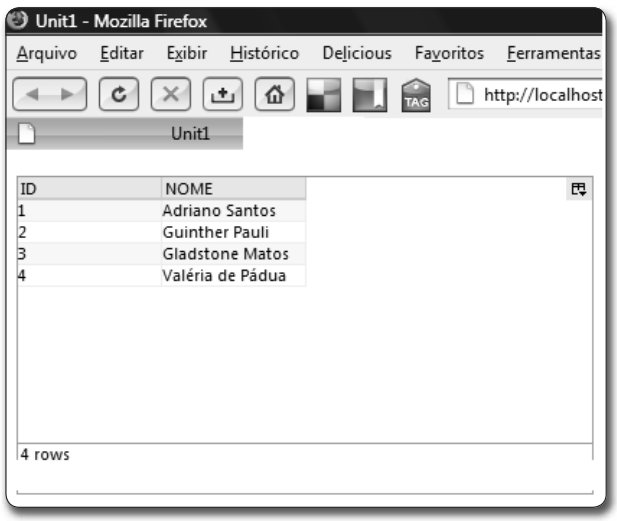


Figura 8. Site em runtime

Listagem 1. Código do evento OnCreate

```
function Unit1Create($sender, $params)
{
  $this->tbclientes1->open();
  $this->tbclientes1->first();
  while (!$this->tbclientes1->EOF){
    $this->cbxClientesGeral->AddItem(
    $this->tbclientes1->NOME, null, $this->tbclientes1->ID);
    $this->tbclientes1->next();
  }
}
```

Se executarmos a aplicação nesse momento, veremos que todos os clientes forma adicionados com sucesso e que podemos retirar o *DbGrid* de tela. Antes de executarmos o teste, vamos fazer com que o *DBGrid Pedido* seja filtrado de acordo com o cliente selecionado. Arraste a tabela *Pedidos* para a tela assim o *Delphi for PHP* criará para nós as conexões necessárias. Em seguida acesse o evento *OnChange* do *ComboBox* e digite o seguinte código.

```
function cbxClientesGeralChange($sender, $params)
{
  $this->tbpedidos1->Filter = 'ID_CLIENTE = '
    . $this->cbxClientesGeral->ItemIndex;
  $this->tbpedidos1->refresh();
}
```

O que estamos fazendo é bem simples. Ajustamos a propriedade *Filter* da tabela *Pedidos* para filtrar, pelo campo *ID_CLIENTE*, usando o *ID* escondido na propriedade *Value* do *ComboBox*. Execute a aplicação e selecione os diversos clientes no *ComboBox*. Repare na barra de status do seu *browser* que há um processamento enquanto os dados são filtrados. Certamente que o teste localmente não vai gerar uma lentidão preocupante, mas imagine sua aplicação na Web com milhares de registros e centenas de pessoas acessando?

Até agora não usamos nada de *Javascript*, muito mesnos *Ajax*. Vejamos no próximo tópico como fazer o uso da tecnologia para agilizar o processo de atualização do *DbGrid* de *Pedidos*.

Agilizando a página

O que fazer agora para que possamos usar *Ajax* em nossa página? A primeira providência a se tomar é observar nas propriedades do formulário que existe uma propriedade chamada *UseAjax*. Essa propriedade indica que a página atual utiliza-se ou não das classes *Ajax* presentes no *Delphi for PHP*. Em outras palavras, precisaremos *seta-la* como *True* para que façamos uso desse mecanismo em nosso *Web Site*.

Outra observação que devemos fazer é que a utilização de *Ajax* no *Delphi for PHP* é feita através do *Framework XAJAX* e com a propriedade *UseAjax* desativada, o sistema não faz menção e inclusão das bibliotecas de uso dessa tecnologia. Experimente executar a página e no *browser* mandar exibir o código-fonte dela. Em seguida ative a propriedade e repita o processo. Verá que uma série de códigos referentes ao XJAX serão adicionados no código-fonte do site como podemos ver na **Listagem 2**.

Listagem 2.Trecho de código com XJAX na página

```
...
var xajaxRequestUri="http://localhost:3569/index.php?restore_
session=1&DBGSESSID=1";
var xajaxDebug=false;
var xajaxStatusMessages=false;
var xajaxWaitCursor=true;
var xajaxDefinedGet=0;
var xajaxDefinedPost=1;
var xajaxLoaded=false;
function xajax_ajaxProcess(){return xajax.call("ajaxProcess",
 arguments, 1);}
</script>
...
```

Agora vem a parte interessante do uso de *Ajax* no *Delphi for PHP*. A verdade é que toda e qualquer programação PHP pode ser chamada via Ajax, ou seja, tanto as funções criadas pelo programador, quanto as criadas pela própria IDE podem ser chamadas tranquilamente via Ajax. E como exatamente fazer isso?

Se observarmos além das propriedades e eventos, há uma aba chamada *Javascript* no *Delphi for PHP*. Isso significa que teremos que copiar o código criado até agora e colar no evento onde queremos que o código seja executado. Ora, o que queremos é que o código do evento *OnChange* do *ComboBox* seja executado como Ajax, e para isso basta que invoquemos o método através do evento *OnChange* da aba *Javascript*. Algumas pessoas podem achar que é só copiar e colar o código, mas na verdade precisamos invocá-lo. Para isso acesse o evento indicado e digite a seguinte instrução:

```
echo $this->cbxClientesGeral->ajaxCall("cbxClientesGeralChange");
```

O método *ajaxCall* envia uma mensagem para a aplicação *dizendo* que esse método precisa ser chamado usando *Ajax*. Repare que passamos como parâmetro o nome da função, *cbxClientesGeralChange*. Se olharmos novamente o código veremos:

```
function cbxClientesGeralChange($sender, $params)
```

Execute a página agora e veja. Com certeza ficará surpreso, que o tempo de carga basicamente ficou idêntico, porém há uma real vantagem em utilizar esse recurso. Experimente voltar no projeto e adicionar alguns *Edit's* na tela. Experimente trocar de cliente e tentar digitar em um dos *edit's*. Você vai notar que ao finalizar o carregamento da página o conteúdo do *Edit* vai voltar ao seu estado inicial. Isso acontece porque precisamos dizer ao método *ajaxCall* que ele deve atualizar somente o *DbGrid* em tela, e não todo o site. E como fazer isso?

O método *ajaxCall* possui mais dois parâmetros. O segundo parâmetro vamos passar em branco, que na verdade é um *array* com os parâmetros passados para a função e o terceiro são os componentes que desejamos que sejam atualizados. Nesse caso somente o *DBGrid*. Nosso código vai ficar como a seguir:

```
echo $this->cbxClientesGeral->ajaxCall("cbxClientesGeralChange",
  array(), array("ddpedidos1"));
```

Execute a aplicação e refaça os testes. Verá que é possível digitar nos demais *edit's* enquanto a página é recarregada. Enfim, a utilização de *Ajax* é realmente muito simples de ser feita no *Delphi for PHP*. Esse artigo na verdade é uma extensão de uma das vídeo-aulas do colunista *Rodrigo Carreiro* sobre *Delphi for PHP*. Se preferir, você pode assistir a essa aula acessando o portal da *DevMedia*. Consulte o quadro *ClubeDelphi Plus* para encontrar o endereço da aula.

Conclusão

Com a onda da Web 2.0 o que mais se ouve falar é em Ajax. Ele agiliza o processamento, organiza a página e ainda permite o multi-processamento no site. Vantagem adquirida somente com o uso dessa tecnologia. Há muito mais o que estudar, então, portanto procure novas informações sobre o tema e melhore a usabilidade de suas aplicações. ●

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora mesmo o portal do assinante ClubeDelphi e veja uma vídeo aula de Rodrigo Mourão como utilizar corretamente o Ajax em aplicações Web.

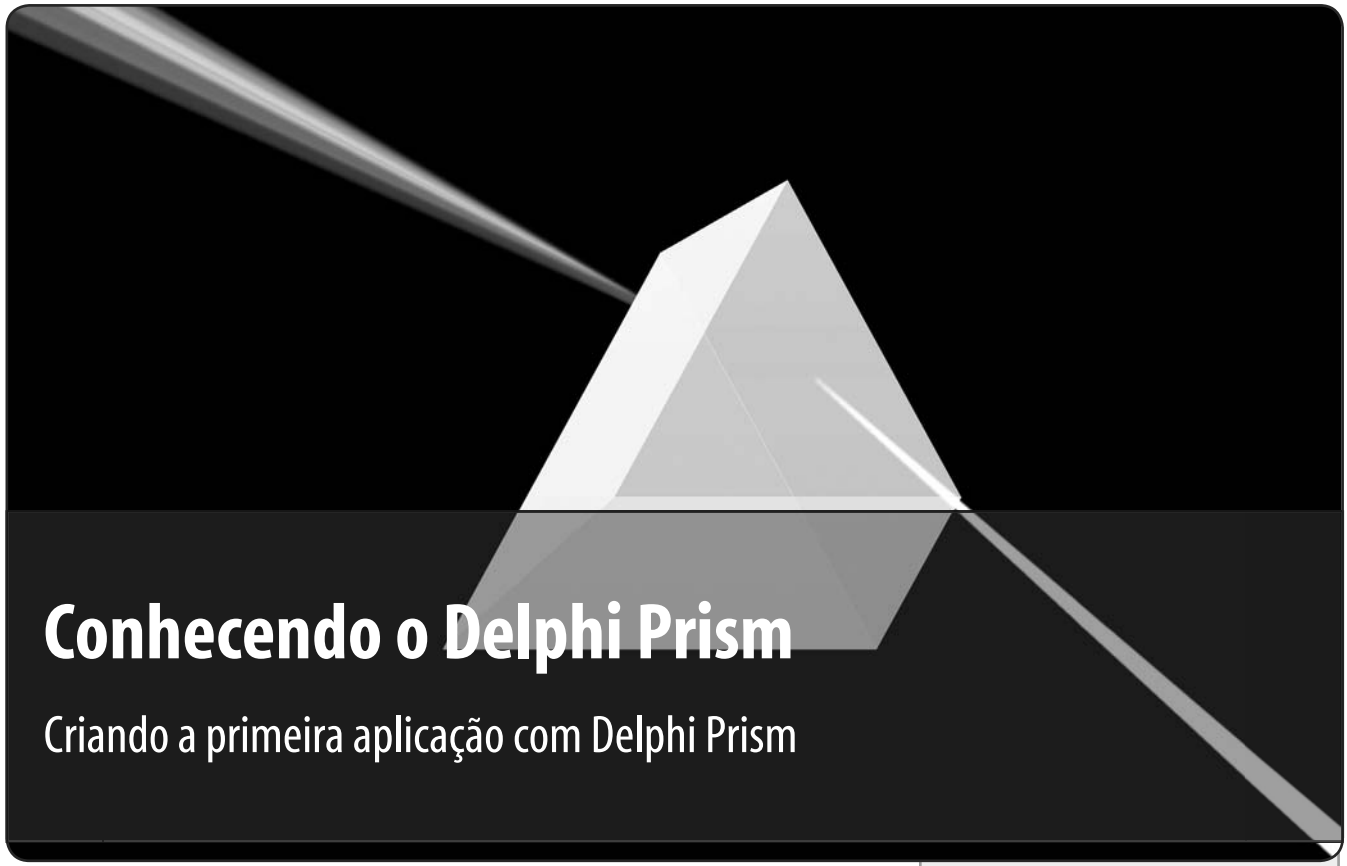
http://www.devmedia.com.br/articles/viewcomp.asp?comp=10971&hl=

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback



Conhecendo o Delphi Prism

Criando a primeira aplicação com Delphi Prism

Nesse artigo veremos
- O que é Delphi Prism;
- Como utilizar o Delphi Prism no dia-a-dia;
- Desenvolvendo um exemplo simples com Prism;

Qual a finalidade
Conhecer a nova IDE desenvolvimento .net da Embarcadero Technologies;

Quais situações utilizam esse recurso?
Com o Delphi Prism é possível criar Web Sites de qualquer porte e distribuí-los para qualquer tipo de cliente e aplicações;

Adriano Santos
falecom@adrianosantos.pro.br
É desenvolvedor Delphi desde 1998, professor e programador PHP, bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É Editor Chefe das revistas ClubeDelphi e WebMobile. Gerente de TI na SoftPark, parceira Borland, membro fundador do DUG-SP (www.dug-sp.com) e mantenedor do blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre o desenvolvimento Delphi.

Resumo do DevMan

O .net tem crescido exponencialmente no mercado de TI nos últimos anos e acompanhar a tendência tornou-se obrigação dos profissionais dessa área. Com o .net sendo cada vez mais utilizado em todas as plataformas ao redor do globo, faz-se necessário utilizar-se dessa tecnologia nos mais variados cenários, visto que atende desde a pequenos até mega-negócios.

A Microsoft tem estimulado o uso do .net e evoluído seu Framework cada vez mais. Até o fechamento dessa edição, a versão 4.0 do Framework estava em testes há bastante tempo já nos laboratórios da empresa.

Para sanar o problema de acompanhamento de tecnologia por parte das mais variadas linguagens de desenvolvimento, a Microsoft criou o Microsoft Visual Studio Shell que veremos mais adiante. Com ele é possível acompanhar a evolução do Framework sem maiores problemas.

A CodeGear, antes sob o comando da Borland, desenvolveu um plug-in para rodar sob o Shell do Visual Studio 2008 o que torna o produto compatível com o Framework mais atual. A Embarcadero Techologies, que adquiriu recentemente a linha de compiladores da Borland, deu continuidade ao projeto e é isso que veremos nesse artigo.

Estamos atualmente na versão *3.5 do Framework .net* da *Microsoft* e acompanhar a evolução dele não é fácil. Até onde sei, são mais de *6.000* programadores dentro da empresa trabalhando nas mais variadas tecnologias de ponta. Até a versão *CodeGear RAD Studio 2007* podíamos desenvolver web sites utilizando a versão 2.0 do Framework. Não que seja ruim, mas como dizem: a fila anda. Novos componentes, novas funcionalidades, novos recursos

são adicionados e a IDE vai ficando para trás sem condições de acompanhar a demanda.

Pensando nisso a *CodeGear* quando ainda comanda pela Borland, desenvolveu um produto capaz de acompanhar esse mercado. O *Delphi Prism* veio justamente com esse propósito, integra-se com o que há de mais moderno.

Em outra linha de frente está a *Microsoft* que, sabendo da necessidade de mercado em atualização e utilização de seu *Framework*, criou o *Microsoft Visual Studio 2008 Shell*. O *Shell* do *Visual Studio* nada mais é do que a própria IDE recebendo plug-ins de novas linguagens, ou seja, é possível desenvolver aplicações que funcionem por baixo do *Visual Studio* e utilizando seus recursos. Isso é interessante, pois abre caminho para que outras empresas do ramo alavanquem seus sistemas e, evidentemente, evoluam com o tempo.

O que veremos nesse artigo é uma introdução ao *Delphi Prism,* criação de uma simples página para entendermos a sua funcionalidade e claro, apresentação da *IDE.*

Baixando e instalando

A primeira providência, claro, é efetuar o download da ferramenta e fazer sua instalação. O requisito básicos para instalação é ter o *.net* instalado no computador, mas caso não tenha, o próprio *Delphi Prism* fará isso para você. Vamos fazer seu download então.

Acesse o link *www.embarcadero.com* e em seguida a opção *Products>Delphi Prism*. No banner principal clique em *Download Trial*. Você será levado a uma tela que possui todas as aplicações relacionadas ao *Prism* para download. Você deverá escolher entre baixar a image (*.ISO*) do arquivo ou simplesmente o instalador on-line. Recomendo que baixe a segunda opção, *CodeGear RAD Studio 2009 ISO Disk 2 – Delphi Prism*. Após efetuado o *download*, inicie a instalação do sistema. Não há segredos, apenas siga as instruções de tela e aguarde.

A instalação é demorada, pois instala o básico para utilização da ferramenta, como o .net Framework, por exemplo.

O *Delphi Prism* utiliza-se do compilador da *REM Objects*, por isso ao abrir o sistema a primeira vez um *splash* (**Figura 1**) aparece fazendo referência a eles.

Conhecendo a IDE

Como mencionado, o *Delphi Prism* na verdade é um plug-in que é instalado sob o *Microsoft Visual Studio 2008 Shell* e, portanto, é dentro dessa IDE que trabalharemos de agora em diante. É importante saber que a linguagem continua a mesma, apenas com algumas alterações para adaptar-se a nova IDE. Um fato interessante é que é totalmente possível utilizar-se de qualquer objeto que o *Visual Studio* possua para incrementar ainda mais sua aplicação. E assim como já falei, o Framework utilizado é o mais recente.

Aberta a IDE, vamos conhecer cada parte do sistema. Na parte superior da IDE, podemos ver os principais comandos para *Abrir, Salvar, Compilar* e etc. No centro

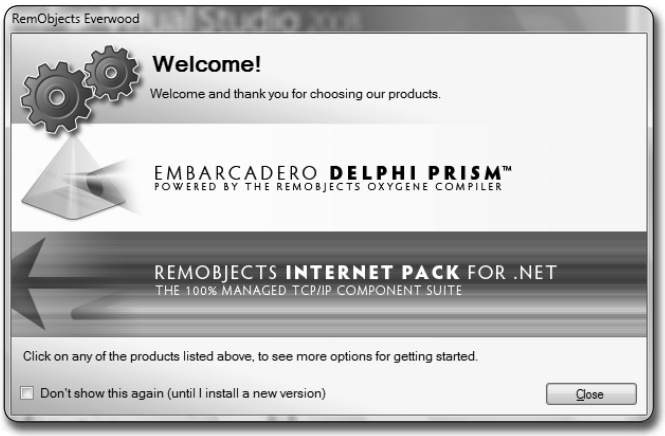


Figura 1. Tela de abertura do Delphi Prism

Figura 2. IDE do Visual Studio Shell

da tela encontramos a *HomePage* que contém atalhos para os projetos recentes e funciona como um *browser* para navegação na *Web*. A esquerda e a direita encontram-se os painéis para as mais variadas operações. O padrão é parecer o *Solution Explorer* na parte direita e *ToolBox* a esquerda do software. Para quem já se utiliza do Delphi 8.0 até a versão 2007, certamente já está acostumando com o layout da IDE, que é bastante semelhante.

Para quem não conhece nada, a *ToolBox* é onde ficam os controles para uso na página e/ou aplicação, assim como a paleta de componentes no *Delphi* tradicional. *Dockado* em *Solution Explorer* e *ToolBox* ainda encontramos *Class View* e *Server Explorer,* respectivamente (**Figura 2**).

Para que possamos visualizar melhor o sistema, vamos criar uma aplicação simples e ver os principais recursos. Para isso clique em *File>Web Site* e perceba que temos a opção de criar três tipos de projetos: *ASP.NET Web Site, ASP.NET Web Service* e *Empty Web Site.* Além disso na parte superior direita é possível escolher qual o *Framework* nós vamos trabalhar. Na parte inferior precisamos escolher o caminho onde os arquivos serão salvos. Escolha o primeiro tipo de projeto o *Framework 3.5*. Se necessário crie uma pasta para armazenar o projeto.

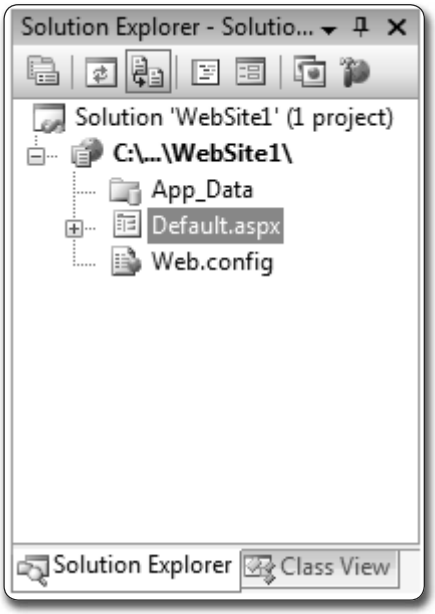


Figura 3. Solution Explorer

Feito isso, perceberá que nosso *Solution Explorer* se modificará mostrando os arquivos do nosso novo site (**Figura 3**). A página principal de nosso site é a *Default.aspx*, perceba que há um sinal de "+" ao lado do arquivo. Isso significa que podemos acessar o código-fonte dessa página, experimente expandir. O arquivo que é mostra tem o nome de *Default.aspx.pas*, ou seja, código pascal. Na página principal de código já podemos ver o código *ASPX* dessa página, para ver o código pascal apenas clique duas vezes no arquivo mencionado. Certamente quem acostumou-se a desenvolver sites usando o *Delphi.NET* nas versões anteriores, não terá grandes dificuldades em se adaptar *Delphi Prism*. Veja nas **Listagens 1** e **2** os códigos *.aspx* e *.pas* da aplicação atual.

Criando uma página simples

Criaremos uma aplicação simples agora para que possamos entender e conhecer os principais recursos da linguagem. O mesmo projeto que foi criado no tópico anterior, já é necessário para que iniciemos nosso teste. Vejamos, crie uma imagem de *600x100px* em formato *JPEG* no *Paint* ou no seu programa preferido. Salve-a como *cabecalho.jpg* no diretório onde salvou a página. Ele servirá de cabeçalho para nossas páginas. No meu caso criei apenas uma imagem com alguns logotipos como vemos na **Figura 4**.

Figura 4. Exemplo de imagem para o site

No *Delphi Prism*, perceba que na página *Default.aspx* há um retângulo levemente contornado por uma linha azul. Para quem nunca trabalhou com *Delphi.NET* ou mesmo páginas em si, esse retângulo é uma *DIV*, uma *tag HTML* que podemos inserir objetos e textos dentro dela. Nesse momento vamos utilizar os objetos da *ToolBox HTML*. O primeiro deles será o objeto *Image*. Clique nele e arraste-o para a *DIV* soltando-o. Um pequeno ícone aparecerá, então selecione na caixa de propriedades a propriedade *Src*. Clique no botão de reticências e então escolha o arquivo cabecalho.jpg no diretório do site. Verá que a imagem é automaticamente carregada para o site e mostrada no ambiente de desenvolvimento.

Agora arraste abaixo do controle *Image* um controle *Horizontal Rule*, para dividir a tela. Diminua o seu tamanho para que fique na mesma largura da imagem e veja se é necessário trocar seu alinhamento para *Left Justify*. Essa opção fica na parte superior direita do *Delphi Prism*. Faremos então a criação de um pequeno menu na parte superior logo abaixo da barra horizontal. Para isso acesse o menu *Table>Insert Table*. Na caixa de diálogo marque uma linha por quatro colunas e 600 pixels de largura. Veja a **Figura 5**.

Digite em cada célula os textos *Principal, História, Clientes* e *Contato*. Clique no botão Source e vamos inserir links para os menus. Localize a *tag Table* e veja log abaixo as *tags TR*

Listagem 1. Código Pascal

```
namespace;

interface

uses
  System,
  System.Data,
  System.Configuration,
  System.Web,
  System.Web.Security,
  System.Web.UI,
  System.Web.UI.WebControls,
  System.Web.UI.WebControls.WebParts,
  System.Web.UI.HtmlControls;

type
  _Default = public partial class(System.Web.UI.Page)
  protected
    method Page_Load(sender: Object; e: EventArgs);
  end;

implementation

method _Default.Page_Load(sender: Object; e: EventArgs);
begin

end;

end.
```

Listagem 2. Código ASPX

```
<%@ Page Language="Oxygene" AutoEventWireup="true"
CodeFile="Default.aspx.pas" Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    </div>
    </form>
</body>
</html>
```

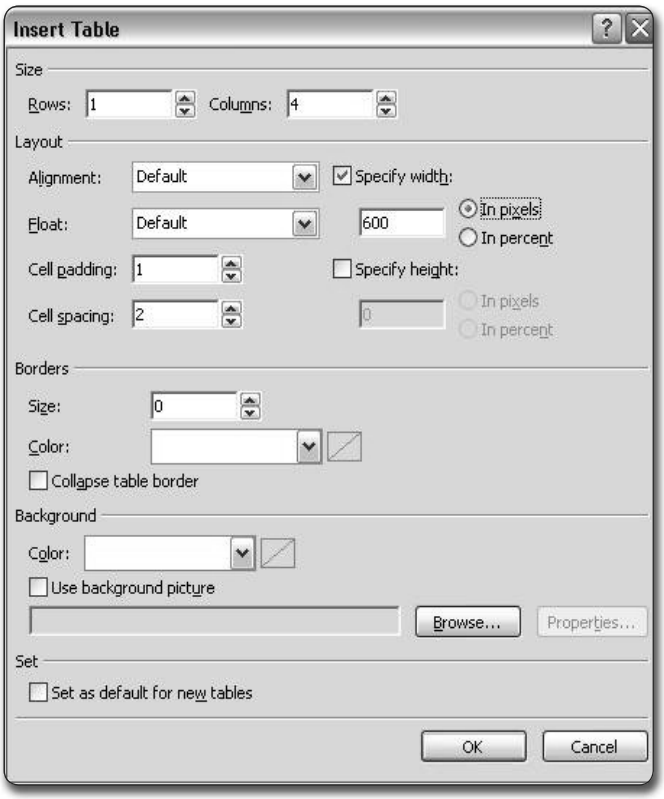


Figura 5. Criação de menu

(*Linha*) *TD* (*célula*). Nas *tags TD* de cada texto, insira uma *tag* <*a href*> apontando para o arquivo que abriremos toda vez que o link for clicado. Ex.:

```
<a href="Default.aspx"><span class="style3">Principal</span></a></td>
```

Veja que no texto *Principal* estamos apontando-o para o arquivo *Default.aspx*. Isso é *HTML* puro, portanto não entrarei em detalhes sobre o tema. Se preferir observe a **Listagem 3** que possui todos os links digitados, inclusive as configurações da tabela. Note também que adicione cor ao fundo das células usando o comando *bgcolor*.

Listagem 3. Configurações da tabela para criação do menu

```
<table class="style1">
  <tr>
    <td class="style4" bgcolor="#3333ff">
      <a href="Default.aspx"><span class="style3">Principal
       </span></a></td>
    <td class="style2" bgcolor="#3333ff">
      <a href="Historia.aspx"><span class="style3">História
       </span></a></td>
    <td class="style2" bgcolor="#3333ff">
      <a href="Clientes.aspx"><span class="style3">Clientes
       </span></a></td>
    <td class="style2" bgcolor="#3333ff">
        <a href="Contato.aspx"><span class="style3">Contato
         </span></a></td>
  </tr>
</table>
```

Agora insira uma nova tabela abaixo de nosso menu, mas dessa vez com apenas uma linha e uma coluna, 600px de largura e alinhamento a esquerda. Esse será o corpo do nosso *web site*. Digite algum texto dentro dele ou copie algo da internet para termos um site perto da realidade. É provável que tenha um site semelhante ao que vemos na **Figura 6**.

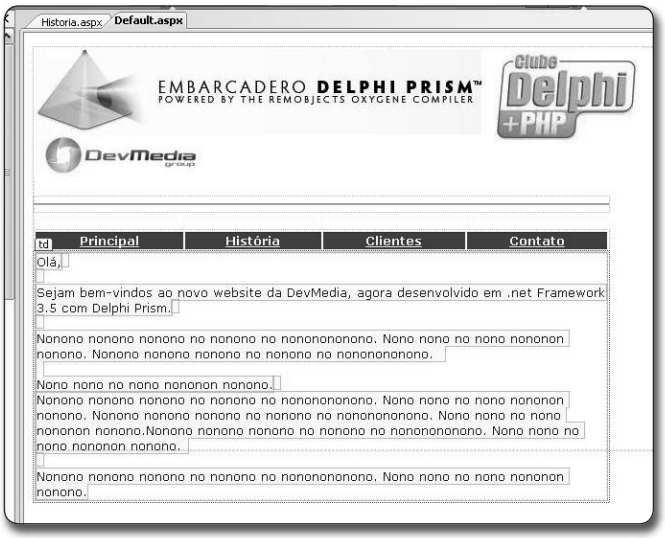


Figura 6. Site em projeto finalizado

Temos ainda somente a página principal, por isso criaremos uma nova página usando botão direito do mouse no *Solution Explorer* e escolhendo a opção *Add new item*. Apenas certifique-se de clicar no nome do projeto para obter o menu desejado. Uma janela com uma série de arquivos será aberta. Nessa tela podemos criar o mais diversos tipos de arquivos para incluir em nossa aplicação, tais como: outras *páginas*, *classes*, *master pages*, *arquivos texto*, etc (**Figura 7**). Selecione *Web Form* e clique em *Ok*.

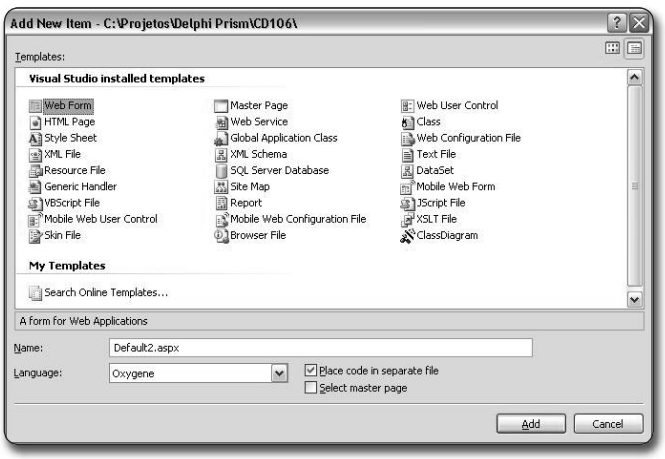


Figura 7. Criando um novo arquivo

Como podemos notar e já era esperado, uma página em branco foi criada. Uma boa dica é copiar todo o conteúdo da *homepage* para o arquivo atual e alterar apenas o necessário. Poderíamos ter usado *master pages* para manter o menu atualizado, mas apenas para fins de conhecimento deixaremos como está.

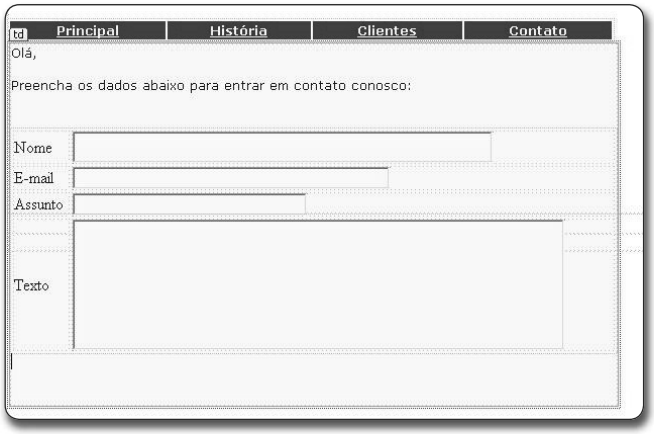


Figura 8. Formulário de contato

Feito isso, execute o site e veja que tudo funciona perfeitamente. Utilizamos até aqui somente os controles da paleta *HTML*. Crie um novo item para que possamos criar um formulário de envio. Criaremos ainda sem funcionalidade, só para que possamos entender os principais controles.

Copie e cole novamente o conteúdo de uma das páginas já pronta para que tenhamos o mesmo layout de site. Em seguida apague o texto do miolo e insira uma tabela de cinco linhas e duas colunas. Nas células da esquerda insira um controle *Label* para cada uma. Modifique a propriedade *Text* de cada um para *Nome, E-mail, Assunto* e *Texto*. Arraste para a direita, um em cada célula, controles *TextBox* presentes na paleta *Standard*.

Pare na última célula da esquerda nessa tabela e arraste para a direta para selecionar a linha toda. Agora clique em *Table>Modify>Merge Cells*. Insira um Button também da última célula. Ao término de tudo seu site se parecerá com a **Figura 8.** Por fim execute o site e veja como ficou. Essa foi apenas uma introdução a nova ferramenta para desenvolvimento *.net* com *Delphi*.

Conclusão

O *Delphi Prism* se mostrou bastante estável em meus testes e acredito que fará muito sucesso entre os desenvolvedores. A parte interessante disso tudo é que é possível manter-se atualizado com o Framework sem necessariamente alterar o tempo todo de IDE de desenvolvimento.

Boa sorte a todos e bons códigos! ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Cursos Online

Assinatura ClubeDelphi PLUS

Mais conteúdo DELPHI por muito menos!

A Revista **Clubedelphi** oferece para seus assinantes uma série de Cursos Online de alto padrão de qualidade .

Conheça abaixo os cursos já disponíveis:
www.devmedia.com.br/curso/cdplus

- **Aplicações Client/Server com dbExpress e Firebird**
- **Sistema SysCash**
- **Criando uma aplicação multi-camadas completa com Delphi**
- **Aplicações client/server com Windows Forms no Delphi 2006**
- **Aplicações WEB com IntraWeb e Delphi 7 (Delphi Win32)**

A sua melhor opção de aprendizagem!

Assine a **Clubedelphi** e
Comece já seu treinamento!
www.devmedia.com.br/assine

DevMedia GROUP Mais Informações: www.devmedia.com.br/central - Tel.: 21 3382-5038/ 3382-5025

***Sr. Cliente favor ler as informações com atenção, dúvidas entrar em contato com atendimento.**

Relatório de Pré-Flight

OP: 65452 Cliente: Dev Midia Data: 30/03/09 Operador: JULIO

Informações da OP:
Título: Clube Delphi
Quantidade de Páginas: 48 + Capa
Formato do Miolo: 205x275
Formato da Capa: 205x275
Tamanho da Lombada: ----------
Tipo de Acabamento: Canoa

Informações do Arquivo:
Título: Clube Delphi
Quantidade de Páginas: 48 + Capa
Formato do Miolo: 205x275
Formato da Capa: 205x275
Tamanho da Lombada: ----------
Tipo de Acabamento: Canoa

Informações do Arquivo

• Imagens em baixa resolução com menos de 200 DPI:
Páginas: TODAS
• Texto preto em quatro cores nas páginas:
Páginas: Capa-04
• Imagens em RGB nas páginas:
Páginas: Capa-02 / Capa-04

• Colofon / Tiragem: ( ) OK ( ) Não
• Código de Barras: (X) OK ( ) Não
• Páginas sem sangria e/ou inferior a 3 mm (Filete branco no acabamento):
Páginas:
• Texto rente ao corte nas páginas:
Páginas:

OBS: As informações em **VERMELHO** necessitam de atenção, se de acordo favor assinar.

Data ____/____/____ Cliente: _____
Nome Legível

Todas as Provas de Cor devem ser assinadas de acordo com a aprovação. Havendo alteração na cor favor anotar, e será enviado nova Prova de Cor.

Observações:

Existem momentos onde a estrutura faz toda a diferença.

Se o tempo é ruim, você encontra na Alog um porto seguro. Isto porque sua empresa reduz custos e melhora a governança em TI. Entre em contato e descubra porque a Alog é líder em hosting gerenciado.

REDUÇÃO DOS CUSTOS AUMENTO DA GOVERNANÇA EM TI

0800 282 3330 | alog.com.br

ALOG
DATA CENTERS DO BRASIL

HOSTING GERENCIADO | COLOCATION | CONTINGÊNCIA | E-MAIL CORPORATIVO | CONECTIVIDADE | SERVIÇOS PROFISSIONAIS

A INVASÃO DE PROPRIEDADE NÃO SE LIMITA A RESIDÊNCIAS.

Assegure-se que seu software está protegido pelo líder em segurança da informação, antes que seja tarde demais.

SafeNet tem à sua disposição a proteção de software mais avançada e segura. As Chaves de Hardware Sentinel oferecem tecnologia que incorpora criptografia de chave pública, criptografia AES e autenticação interna para garantir o mais alto nível de proteção anti-pirataria.

Peça seu Kit de Desenvolvimento hoje mesmo.

Entre em contato em +55 11 4208-7700 em SP-Brasil
ou por e-mail infopirataria@safenet-inc.com

25 YEARS
1983–2008

SafeNet